Hadoop class Room Notes

Call Now: 9989134470

Classroom & Online Training on Apache Hadoop at Q Edge Technologies

No One can Teach Apache Hadoop Training of multiple concepts and modules like SQOOP, HDFS, PIG, Map-Reduce, HBASE, HIVE, and ZOOKEEPER etc, Except Q Edge Technologies.

"Before joined here I attended 4 Demos in other so called experts in Apache Hadoop training ,but nobody convince me better then this institute.I am happy about my decision.I am in MNC now.Choose right training centre before joined."

– Mahir Alfahal

Book an Appointment

Service *	Select a service
	Select desired service
Date	
	Select from available dates
Time *	Select Time
	Select from available time slots
Email	
	Notifications will be sent to this email

http://hadooptraininginhyderabad.co.in/
Call for Demo on +91 9989134470

http://hadooptraininginhyderabad.co.in

Hadoop is a frame work to store and process big data. If the amount of data is more than tera bytes then such a data is called as big data.

Or

The amount of data that is beyond the storage and processing capabilities of single physical machine then it is called as big data.

Hadoop can process

1.Unstructured data

ex: like files, images ,sentimental data which is maintained by networking sites.

2.semistructured data like

ex: XML files, Excel work sheets..

3.Structured data

ex: like tables information which is present in RDBMS.

RDBMS systems can process maximum tera bytes of data only. There are not designed to process big data.

In case of RDBMS to process huge amount of data then we need specialised softwares and high configured hardware(expensive).We can overcome the above problems using hadoop.it is completely free downloaded (open source).To work with hadoop just we need to maintained commodity systems(i.e we need not use high configure hardware).

Advantages of Hadoop :

1.Scalability.

3.fault tolerance.

4.High Availability.

Hadoop was developed by "doug cutting"

Hadoop is not a technology name but it is a "toy name".

Hadoop consist of

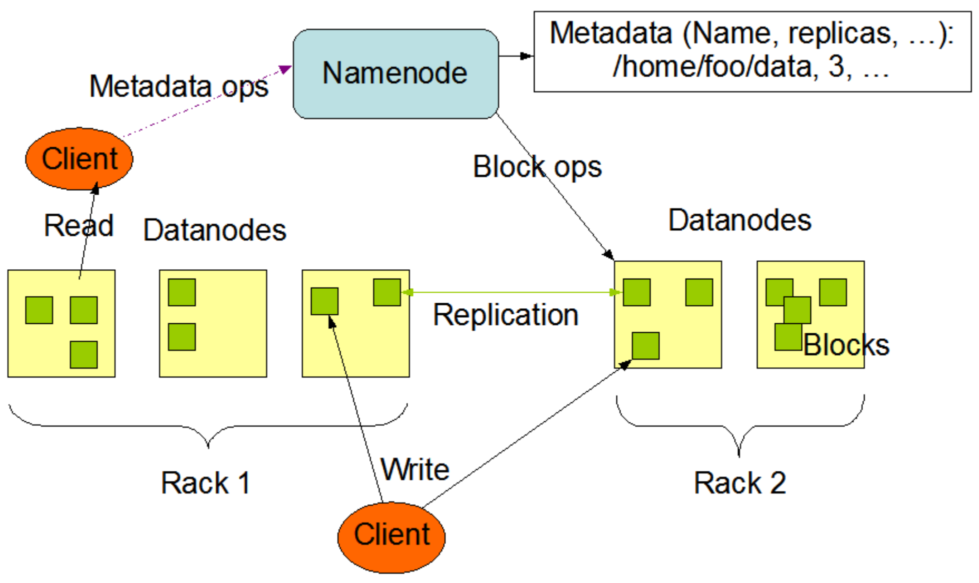
1. HDFS(Hadoop Distributed File System which is implemented in java)
2. Mapreduce.(it is a programming module to process data present in hdfs);

HDFS:

It is also called as master slave architecture.

In Hadoop cluster there will be one main node called NameNode(Master).The other nodes which are connected to NameNode are called DataNodes(slaves).At the time of copying the file into HDFS HdfsClient(bulit in framework classes)receives receives command forwards it to the NameNode.NameNode checks whether the specified file is already existed or not and also checks whether user having sufficient privileges or not .If the file is already existed (or) user not having sufficient privileges ,then it will display an I/O Exception. If the specified file is not existed then the entire file wil be divided into smaller chunks. Each chunk is called a block. Default size of each block is 64 mb.But it can be configured and we can increase the size. The maximum suggested size of a each block is 128 mb.

HDFS Architecture



NameNode will decide where exactly block has to be stored. If the block1 is stored in DN1 then the block2 can be stored in any other datanode .

Replication policy:

Incase of hadoop by default each block is stored 3 times in the hadoop cluster .i.e every file is stored 3 times in the hadoop cluster. this is called Replication policy.

Through configuration files we can increase or decrease replication factor value .

http://hadooptraininginhyderabad.co.in/ http://hadooptraininginhyderabad.co.in

NOTE: Two continuous blocks of same files will not be stored in same data node But more than one block of same file can be stored in same data node.

Name node :

NameNode is the masters node which is responsible to maintain meta data information about each file which is present in HDFS . i.e name node maintains

1. path of the file
2. number of blocks which are available for each file
3. Address of data nodes where exactly each block is present.

The above information is stored in FSImage file & Editlog file.

Without NameNode the file system cannot be used. If name node goes down there is no way of reconstructing the files from the blocks on the data nodes. So NameNode is the single point of failure in generation one hadoop.In generation one hadoop there is secondary namenode. When the name node starts up, it merges the fsimage and edit log files to provide up to date view of the file system metadata. The namenode then overwrites fsimage with the new HDFS state and begins a new edit log.

The checkpoint node periodically creates checkpoints of the namespace. It downloads fsimage and edits from the active namenode. Then it merges locally and uploads the new image back to the active namenode.

The checkpoint node usually runs on a different machine than the namenode as its memory requirements are on the same order as the namenode.it keeps a copy of the merged namespace image which can be used in the event of the namenode failing.

In generation two hadoop there are two master nodes in .if active name node goes down then automatically passive namenode becomes active namenode.

The big companies like google,Facebook maintain clusters in the form of of DataCenters. Data centers usually be in defferent locations like US,Canada,UK etc... Data center is a collection of Racks(rack1,rack2,.....). A rack is nothing but set of nodes (machines).These node are connected through n/w and each data centre is also connected through n/w .This entire set up is called n/w topology.

Note : The main idea of replication polocy ,is if the first block is stored in rack1 of data centre 1 then the second block can be stored in anothers rack of same data center or some other data centre.

CAP Theorem :

C : Consistancy

A : Availability

P : Fault Tolerance

<u>Consistancy :</u>

Once we write any thing into the hadoop cluster we can read it back at any movement of time without loss of data. Hadoop supports 100% consistency.

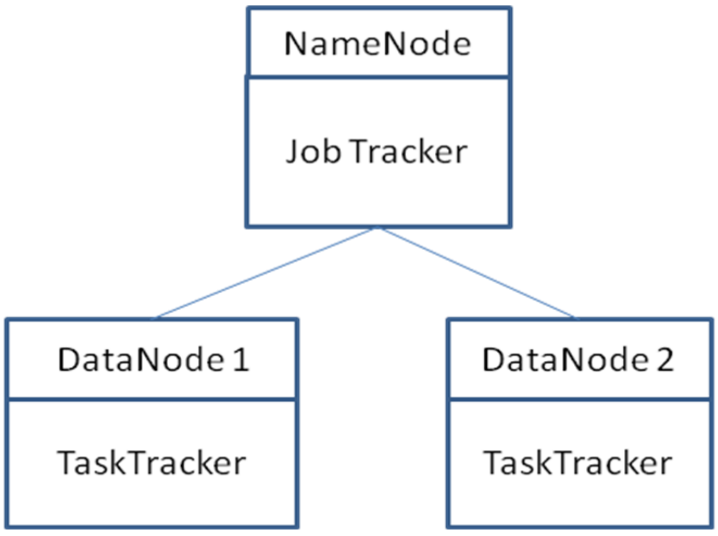
<u>Availability :</u>

There is no chance of failure (approximately 99 % available)

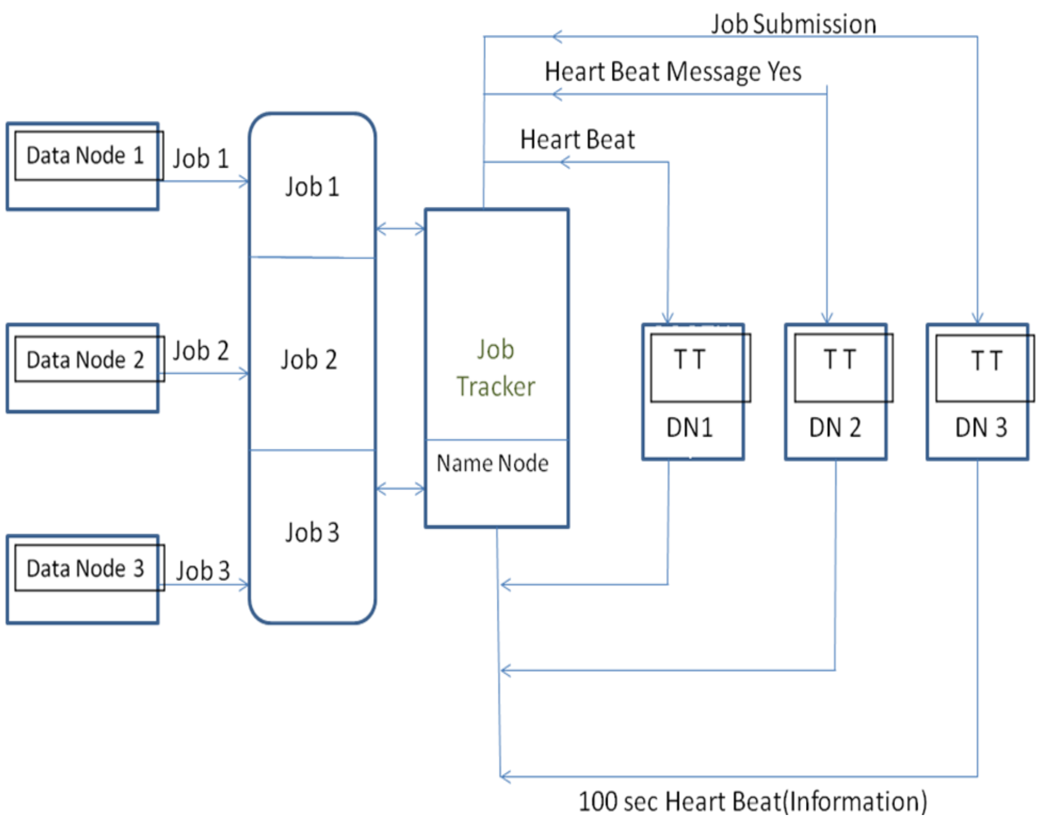
<u>Fault Tolerance :</u>

Even there is n/w break downs between two data centres or any two racks or any two nodes still we are able to process data which is present in clusters.

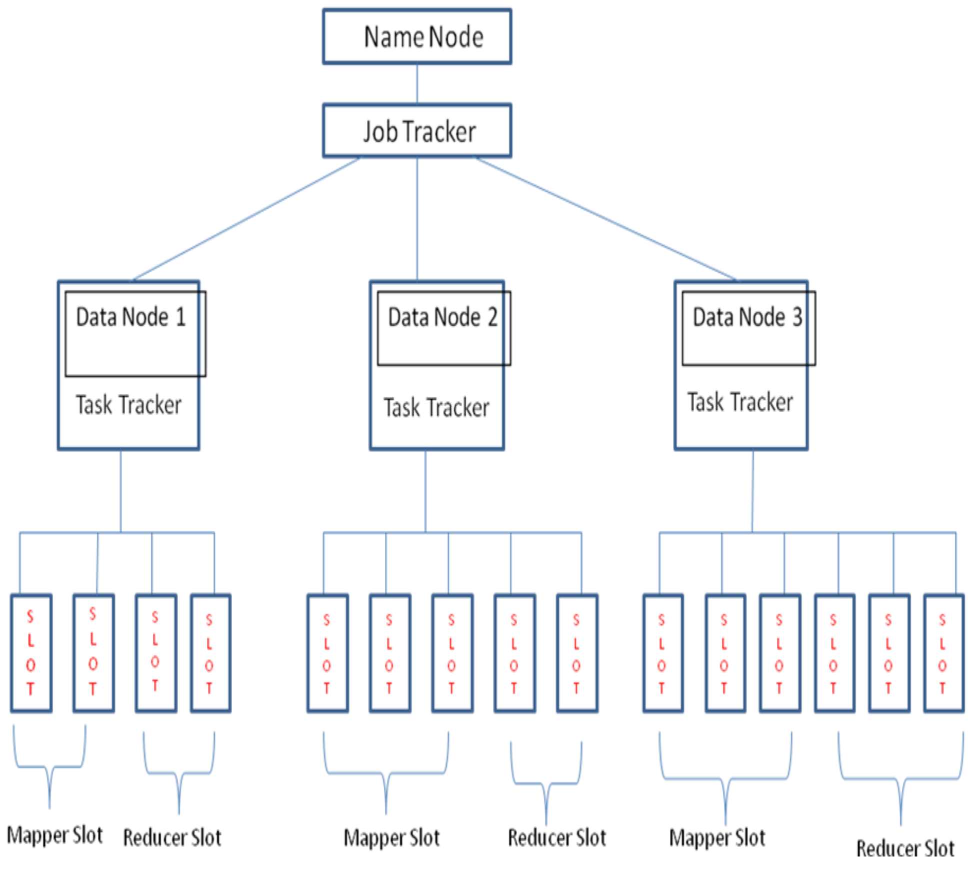
<u>MAP REDUCE</u>



Map reduce is a programming model for data processing. It is also called as master & slave architecture.Job tracker is the master process in mapreduce where as task tracker is the slave in Map Reduce. These are the processes which are used to process the data present in HDFS.Job tracker is running under name node machine where as task trackers run on data nodes.



Before the JobTracker chooses a task for the TaskTracker, the JobTracker must choose job from the list of jobs which are submitted by multiple users from multiple data nodes. Job tracker selects highest priority job from the list(user can assign priority to job while submission to the cluster).The job tracker chooses a task from the job & assign it to the TaskTrackers. The job tracker coordinates all the jobs assigned them to task trackers. TaskTrackers run the job and send heart beats(program reports) to the JobTracker. JobTrackers run a simple loop that periodically sends heart beat method calls to the JobTracker. Heart beats tell the job tracker that a task tracker is alive. As a part of the heart beat task tracker will indicate whether it is ready to run a new task. If it is able to start a new task then JobTracker will allocate a new task to the TaskTracker.JobTracker keeps over all progress of each job .If the task fails then the job tracker can reschedule it on another task tracker.

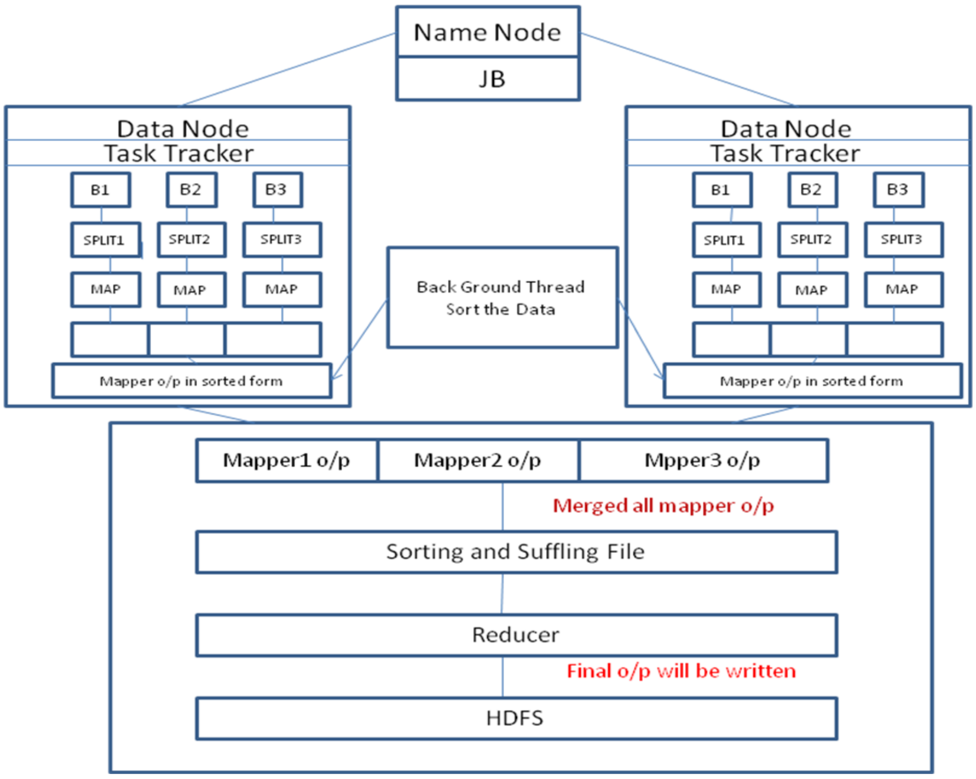


Task Tracker have a fixed number of slots for map tasks and for reduce tasks.These are set indenpendently .

For ex:

A Task Tracker may be configured to run two map tasks and two reducer tasks simultaneously (number of tasks depends on amount of memory on the Task Tracker system. In the context of given job the default scheduler fills empty map tasks slots before reduce task slots. So if the TaskTracker has atleast one empty map task slot ,the Job Tracker will select a map task otherwise it will select a reduce task.

http://hadooptraininginhyderabad.co.in

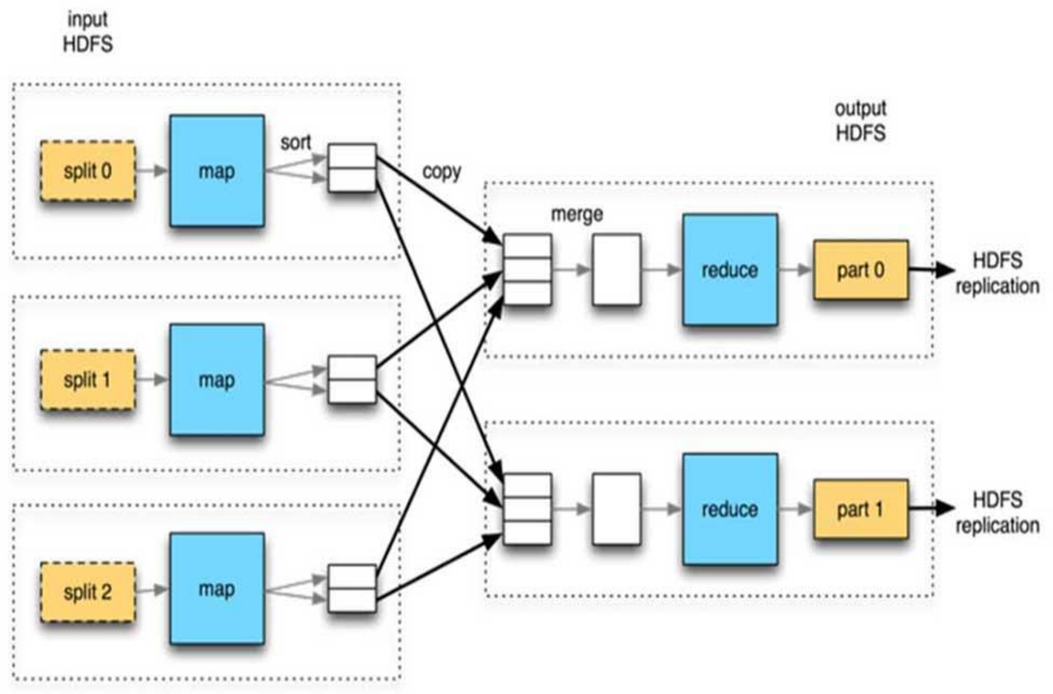


Hadoop frame work divides input of the map reducer into fixed size pieces called input splits. Hadoop creates one map task for each split. Each split will be divided into records(every row is a record). For every record one unique number will be assigned. This number is called offset code. For each record in the split user defined function will be that function name is map(). Having many splits means the time to process each split is smaller compared to the time to process the whole input. For most of the jobs a good split size is 64mb by default. Hadoop creates map task on the data node where the input resides in the Hdfs. So, this is called data locality optimization.

Map tasks write their O/p to the local disks not to Hdfs. Map output is intermediate output. Again it is processed by reduces to provide final output. Once the job is completed the mapper output can be thrown away. If the node is running and map task fails before the mapper output has been consumed to reduced task then the job tracer will automatically create map task on another node.

The sorted mapper outputs have to be transferred across the n/w to the node where the reduce task is running. Then they are merged and passed to the user defined reducer function. Output of the reducer is normally stored in Hdfs for the reliability.

http://hadooptraininginhyderabad.co.in

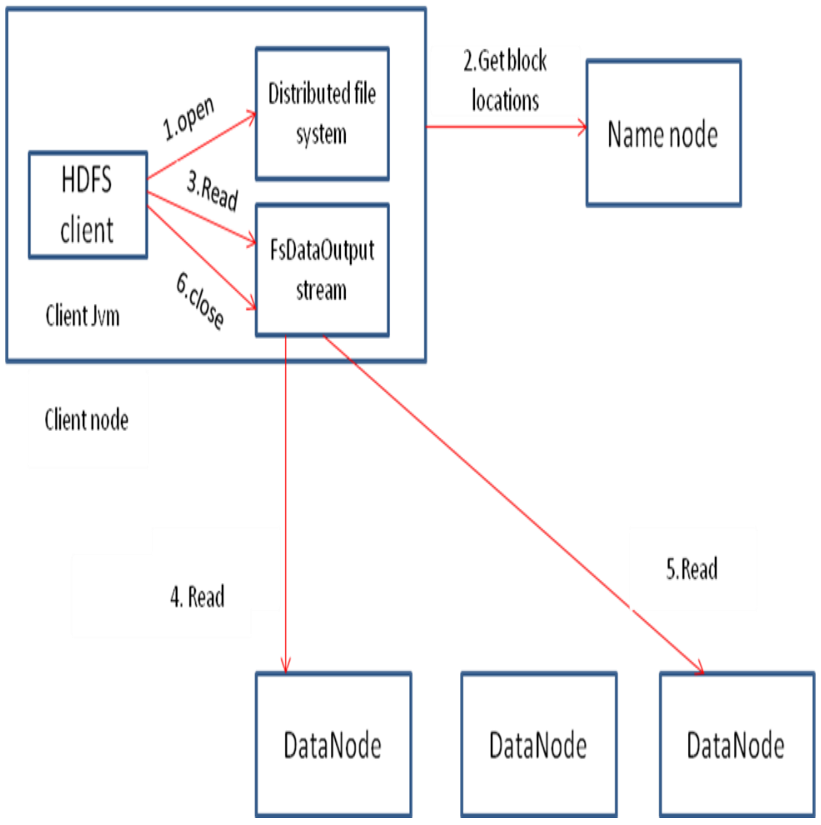


The input to a single reduce task is normally the output from all mappers. In the present example, we have a single reduce task that is fed by all of the map tasks. Therefore the sorted map outputs have to be transferred across the network to the node where the reduce task is running, where they are merged and then passed to the user-defined reduce function. The output of the reduce is normally stored in HDFS for reliability.

The number of reduce tasks is not governed by the size of the input, but is specified independently. In "The Default MapReduce Job" , you will see how to choose the number of reduce tasks for a given job.When there are multiple reducers, the map tasks *partition* their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for every key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner which buckets keys using a hash function works very well. it's also possible to have zero reduce tasks.

Combiner Functions

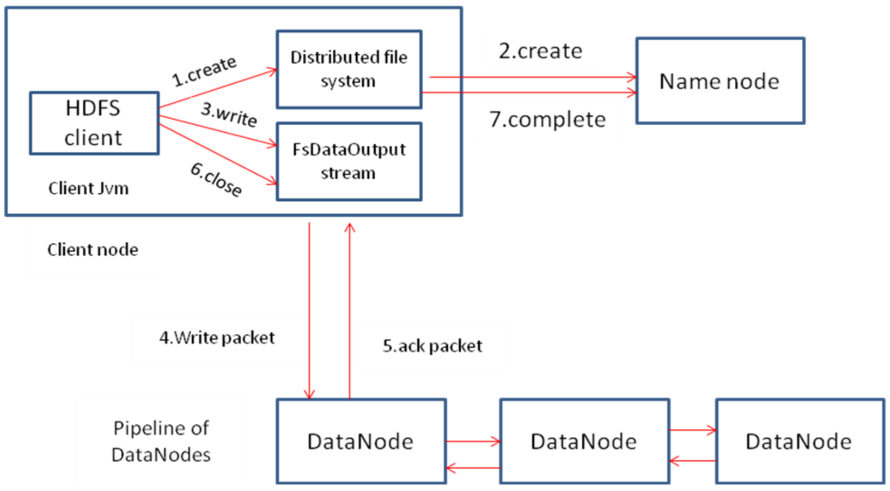
Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a *combiner function* to be run on the map output—the combiner function's output forms the input to the reduce function. Since the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer.



Anatomy of a File Read:

The client opens the file it wishes to read by calling open() on the File System object,which for HDFS is an instance of DistributedFileSystem .Distributed FileSystem calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2). For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client (according to the topology of the cluster's network; see "Network Topology and Hadoop" on page 64). If the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode.The DistributedFileSystem returns a FSDataInputStream (an input stream that supports file seeks) to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the datanode and namenode I/O.The client then calls read() on the stream (step 3).

DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first (closest) datanode for the first block in the file. Data is streamed from the datanode back to the client, which calls read() repeatedly on the stream (step 4). When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block (step 5). This happens transparently to the client, which from its point of view is just reading a continuous stream. Blocks are read in order with the DFSInputStream opening new connections to datanodes as the client reads through the stream. It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close() on the FSDataInputStream (step 6). During reading, if the client encounters an error while communicating with a datanode, then it will try the next closest one for that block. It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks. The client also verifies checksums for the data transferred to it from the datanode. If a corrupted block is found, it is reported to the namenode, before the client attempts to read a replica of the block from another datanode. One important aspect of this design is that the client contacts datanodes directly to retrieve data, and is guided by the namenode to the best datanode for each block.



Anatomy of a File Write:

The client creates the file by calling create() on DistributedFileSystem (step 1 in Figure 3-3). DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it (step 2). The namenode performs various checks to make sure the file doesn't already exist, and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file

creation fails and the client is thrown an IOException. The DistributedFileSystem returns a FSDataOutputStream for the client to
start writing data to. Just as in the read case, FSDataOutputStream wraps a DFSOutput Stream, which handles communication with the datanodes and namenode. As the client writes data (step 3), DFSOutputStream splits it into packets, which it writes to an internal queue, called the data queue. The data queue is consumed by the Data Streamer, whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline—we'll assume the replication level is 3, so there are three nodes in the pipeline. The DataStreamer streams the packets to the first datanode in the pipeline, which stores The Hadoop Distributed Filesystemthe packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline (step 4). DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the ack queue. A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline (step 5).

If a datanode fails while data is being written to it, then the following actions are taken,
which are transparent to the client writing the data. First the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets. The current block on the good datanodes is given a new identity, which is communicated to the namenode, so that the partial block on the failed datanode will be deleted if the failed datanode recovers later on. The failed datanode is removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline. The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. Subsequent blocks are then treated as normal.

It's possible, but unlikely, that multiple datanodes fail while a block is being written. As long as dfs.replication.min replicas (default one) are written the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication factor is reached dfs.replication, which defaults to three). When the client has finished writing data it calls close() on the stream (step 6). This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step 7). The namenode already knows which blocks the file is made up of (via Data Streamer asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

Replica Placement How does the namenode choose which datanodes to store replicas on? There's a tradeoff between reliability and write bandwidth and read bandwidth here. For example, placing all replicas on a single node incurs the lowest write bandwidth penalty since the replication pipeline runs on a single node, but this offers no real redundancy (if the node fails, the data for that block is lost). Also, the read bandwidth is high for off-rack reads. At the other extreme, placing replicas in different data centers may maximize redundancy, but at the cost of bandwidth. Even in the same data center (which is what all Hadoop clusters to date have run in), there are a variety of placement strategies. Indeed, Hadoop changed its placement strategy in release 0.17.0 to one that helps keep
a fairly even distribution of blocks across the cluster. (See "balancer" on page 284 for details on keeping a cluster balanced.) Hadoop's strategy is to place the first replica on the same node as the client (for clients running outside the cluster, a node is chosen at random, although the system tries not
to pick nodes that are too full or too busy).

The second replica is placed on a different Data Flow | 67rack from the first (off-rack), chosen at random. The third replica is placed on the same rack as the second, but on a different node chosen at random. Further replicas are placed on random nodes on the cluster, although the system tries to avoid placing too many replicas on the same rack. Once the replica locations have been chosen, a pipeline is built, taking network topology into account. For a replication factor of 3, the pipeline might look like Figure 3-4. Overall, this strategy gives a good balance between reliability (blocks are stored on two racks), write bandwidth (writes only have to traverse a single network switch), read
performance (there's a choice of two racks to read from), and block distribution across the cluster (clients only write a single block on the local rack)

<u>OOP</u>

Object:- Object is defined by state and behaviour. State represents physical appearance of an object. Behaviour represents purpose of an object. In case of oop ,state is nothing but application data. It can be maintained through member variables. Behaviour can be implemented through member functions.

Syntax to Create an object.

```
Class classname
{
Access specifier datatype 1 var1, var2.......
Access specifier datatype2  var1,var2........
.
.
}
Access specifier  returntype function1(parm)
{
------
------
}
Access specifier  returntype function2(parm)
{
------
------
}
```

Class:-

Class is a keyword used to bind variables and methods to single unit.

Access Specifiers:-

Used to set the scope of the variables. There are 3 types of access specifiers. public,private and protected . If any member is preceded by any public specifier then it can be accessed through out java programming. Every program requires one entry point. Then only end users can access class properties so this entry point should be public.

Private:-

If any member is preceded by private then it can be accessed with in the class. But not outside of the class.

All the sensitive data like username password etc must be declared under private section.

Book lib collection of racks→rack is a collection of books →book is a collection of pages →page is a collection of info

Java lib is a collection of packages→ package is collection of classes → class is a collection of methods → method is collection of instructions

Program

```
Class testempmaster
{
public static void main (string [] args)
{
empmaster e1,e2;
e1= new empmaster();
e1.eid=101;
e1.ename="ajay";
e1.ebasic=25000.60f;
e2=e1;
System.out.println("eid="+e2.eid);
s.o.p("ename="+e1.ename);
s.o.p("basicsal="+e1.ebasic);
}
}
```

```
class empmaster
   {
 int eid;
      string ename;
   float basic;
}
```

Constructor:

1)constructor having same of the class name

2)it is very similar to function

3)it has no return type

4)it is used to assign initial values to the member variables.

5)it is called by java run time at the time of object creation.

Ex Program:

```
Class empmaster
{
int  eid;
String ename;
float basic;
empmaster()
{
eid=101;
ename="ajay";
basic=20000.50f;
}
empmaster(int id,String name, float sal)
{
eid=id;
ename=name;
basic=sal;
}
void displayemp()
{
s.o.p("empid="+eid);
s.o.p("empname="+ename);
s.o.p("ebasic="+basic);
}
}
class testempmaster
{
public static void main(String [] args)
{
empmaster e1,e2,e3;
e1=new empmaster();
e2=new empmaster(102,"ajay",3000.50);
e1.displayemp();
e2.displayemp();
}
}
```

Inheritance

It is a process of creating new class called derived class from the existing class called base class.

Advantages:

1.code reusability

2.extendability

3.Reliability

4.better maintenance

syntax:

```
Class baseclass
{
Member Variables
+
Member functions
}
Class subclass name extends baseclass name
{
Member Variables
+
Member functions
}
```

Types of inheritance

1. Single inheritance
2. Multi level inheritance
3. Hierarchal inheritance
4. Multiple inheritance
5. Hybrid inheritance

1.single inheritance:

If any class is derived from only one base class the such a inheritance is know as single inheritance

Ex:

```
Class c1
{
Int a,b;
}
Class c2 extends c1
{
Int c,d;
}
C1 obj1=new c1();
C2 obj2=newc2();
```

2.multi level inheritance:

If there is more than one level of inheritance then such a type of inheritance is called multilevel inheritance

Ex:

```
Class c1
{
Int a,b;
}
```

```
Class c2 extends c1
{
Int c,d:
}
Class c3 extends c2
{
Int f,g;
}
C1 obj1 = new c1();
C2 obj2 = new c2();
C3 obj3 = new c3();
```

3.<u>hierarical inheritance</u>:

If more than one class is derived from single base class is called hierarchal inheritance

```
C1
C2          c3          c4
Class c1
{
Int a,b;
}
Class c2 extends c1
{
Int c,d:
}
Class c3 extends c2
{
Int  e,f;
}
Class c4 extends c1
{
Int g,f;
}
C1   obj1 = new c1 ();
C2   obj2 = new c2 ();
C3   obj3 = new c3 ();
C4   obj4 = new c4 ();
```

<u>Function overriding</u>:

If any method is implemented in both base class and derived class with same name and same signature then the method in the derived class is said to be overridden method of super class

Ex:

```
class bc
{
int i,j;
public bc(int a,int b)
{
i=a;
j=b;
}
public  void show()
{
s.o.p("i="+i);
s.o.p("j="+j);
}
}
class dc extends bc
{
int k;
public  dc(int x, int y ,int z)
{
super(x,y);
k=z;
}
public void show()
{
s.o.p("k="+k);
}
}
```

<u>Super</u>:

it is one built in keyword. It contains the super class object reference

<u>advantages</u>:

1. Using super keyword we can call super class constructor from the derived class constructor
2. We can call super class method from the sub class
3. We can access super class member variable from the sub class

<u>This</u>:

It is also a built in keyword it contains the current class object references

<u>Advantages</u>:

- Using "this" we can access member variables of a class and we can call member functions of a class
- In the above example the following statement will make a call to show method of "dc" class only obj.show();

<u>Binding</u>:

It is the process of creating a link between function calling and function definition

There are 2 types of binding

1.static binding

2.dynamic binding

1.<u>static binding</u>:

In the above example link is created between obj.show() and its definition during compilation only

2.<u>dynamic binding</u>:

Rules to implement dynamic

1. Both the sub class and base class must have same function name and same signature
2. Base class methods must be abstract
3. Both the base class method and derived class methods must be called with respect to reference variables
4. Before calling a particular class method using base class reference variable it must have appropriate object reference

Syntax:

Access specifier abstract returntype functionname(parameters)

Ex:

```
Public abstract void area();
Public abstract void peri();
```

Syntax:

```
Access specifier abstract class classname
{
Member variables
+
Implemented methods
+
Abstract methods
}
Ex:
abstract class shape
{
float a
public void getdata()
{
}
public abstract void area();
public abstract void peri();
}
shape S;//no error
// s = new shape();//error
class circle extends shape
{
public void main area()
{
}
public void peri()
{
}
}
```

If any method has no definition such a method is called **abstract** method

If any class having one abstract method then such a class is called abstract class

We can create reference variables of type abstract classes

Ex;

shape   s;

We cannot create object of type abstract classes

Ex:

s=new shape ();

Every abstract classes requires sub class sub class can inherit implemented methods and abstract methods also From the sub class we must override all the abstract methods which are inherited from the abstract base classs other wise derived class becomes base class

Ex:

```
class drinkable
{
public abstract void drink();
}
class tea extends drinkable
{
public void drink()
{
s.o.p("here is your drink");
}
}
class coffee extends drinkables
{
public void drink()
{
s.o.p("here is your coffee");
}
}
class softdrink extends drinkables
{
public void drink()
{
s.o.p("here is your softdrink");
}
}
class drinkabledemo
{
public static void main(string[] arg)
{
```

```
Drinkable d;
int token=integer.parseint(args[0])
switch(token)
{
case 1: d= new tea();
break;
case 2 : d =new coffee();
break;
case 3: d = new softdrinks();
break;
default:
s.o.p("invalid choice");
}
d.drink();
}
}
```

Save it:

```
C:\mat\drinkabledemo.java
C:\mat\javac drinkabledemo.java
C:\mat\java drinkabledemo
```

Interface:

Interface are syntactically very similar to classes

Interface is a keyword used to create interface

Syntax:

```
interface interface name
{
Variables
+
Methods without body
}
```

By default all the interface variables are final and static

Final is a one built in keyword it is used to create constants

if the variable is static we can access it with respect to interface name

By default all the interface methods are abstract methods they are by default public

Ex:

```
interface inf1
{
int x=5000;
void f1();
void f2();
}
```

Reference variables of type interface can be created

Ex: inf1 i;//no error

Object of type interface cannot be created

// i = new inf1(); //error

Every interface requires subclass from the subclass we must override all the dummy or duplicate methods of an interface which are inherited

```
class demo implements inf1
{
public void f1();
{
}
public void f2();
{
}
```

Ex:

```
interface animal
{
void move();
}
class implements animal
{
public void move()
{
s.o.p("cat move");l
}
class dog implements animal
{
public void move()
{
s.o.p("dog move");
}
public void bark()
```

```
{
s.o.p("dog bark");
}
}
class animaldemo
{
public static void main(string  args[] )
{
Animal a;
String s= args[0];
If (s.equalsignorecase("cat")
a=new cat();
a.move();
if (a instanceof  dog)
{
// a.bark() //error
dog d = (dog) a
d.bark();
}
}
```

Static – type of reference variable

Dynamic – runtime value present in refer

Strings:

In java strings is a object of type string class or string Buffer class.

String object contents are not modifiable (immortable)

String Buffer object contents are modifiable

Contructor of String class:

1. String s1=new String()

s1=''Hello'';

s.o.pln(s1)

2. String s2=new String(''Hello'');

s.o.pln(s2);

o/p:-Hello.

3   Char[ ]x={'I' , ' ' , 'L' ,'i' ,'k' ,'e' ,' ''j' ,'a' ,'v','a'}

String s3=new String(x)

s.o.pln(s3);

o/p:-I like java

4   byte [ ]b={65,66,67,68,69,70}

String s4=new String (b)

s.o.pin (s4);

o/p:-A,B,C,D,E,F

Methods:

1.Public int length( )

ex:-String 3=''Hello'';

s.o.pln(s.length( ))

o/p:-5

2.Public char chartAt(int index)

ex:-string s=''Hadoop'' ;

s.o.p (3.charAt(2))

o/p:-d

3.public boolean equals (string obj)

ex:-string s1=''ABC''

string s2=''abc''

s.o.p (s1.equals (s2));

o/p:-32

In the above case equals method takes binary comarision. But here ABC is not equal to abc

4.Public booiean equalsIgnoreCase(String obj)

ex:-sop (s1.equalIgnoreCase (s2))

o/p:-True

5.Public string concat (string obj)

ex:-string s1=''Hello''

string s2=''Ajay''

string s3=s1.concat (s2)

s.o.p (s3);

o/p:- Hello Ajay

6. Public int compareTo (string obj)

This method returns either

1) >0(string 1>string 2)
2) < 0(string 1<string 2)

3) = = 0(string1==string 2).

ex:-string s1=''ABC ''

string s2 =''abc''

int x =s1.compare To (s2)

if (x>0)

s.o.p (''s1>s2'')

else if (x<0)

s.o.p ('' s1<s2'')

else

s.o.p (''s1= =s2'').

Public string trim ( )

ex:-string s1 =''-----Hello-----''

s.o.p (s1.length ( ))

o/p:-5

string s2=s1.trim ( )

s.o.p (s2.length ( ))

o/p:-5

s.o.p ((s1.trim( )).Length ( ))

o/p:-5

Public string substring (int start index ,int end index)

string s ='' I Like hadoop ''

s.o.p (s.substring (2,6))

o/p:-Like

StringBuffer:-

Constructors of StringBuffer:-

StringBuffer sb=new StringBuffer ( )

16 locatations

sb =''Hadoop''

s.o.p (sb);

o/p:-Hadoop

For the above objects system allocates 16 contineous locations.

2.StringBuffer Sb = new StringBuffer("Hello");

21 Locations

(5+16)

Length of string+16 extra locations

3.Public string Buffer Insert (int index, String sub string)

Eg:StringBuffer Sb = new SringBuffer (" I   Hadoop ")

sb.insert (1, "like")

O/P : I like Hadoop.

4. Public StringBuffer append (String obj)

Eg: StringBuffer Sb = new StringBuffer (" Hello ")

sb. append ("world")

----> String s= "hyd is capital of AP"

Public Boolean Contains (String obj)

S.O.P (S.Contains ("capital"));

O/P :   True.

5.Public String replaceAll (string s1,String s2)

String S1 = S.replace All ("Hyd", "Sec")

S.O.P (S1)

O/P : sec is capital of a.p

StringTokenizer :

It is a built in class which is present under java. util package.It is used to break the text in to different parts.Each part is called token. A Token is a valid word.

Constractors of StringTokenizer:

1. StringTokenizer(String to be Tokenized)

Eg : StringTokenizer st=new StringTokenizer("I like Hadoop");

In the above constractor system uses white spaces(space,\n, \t)to break the text.

2 . StringTokenizer (String to be Tokenized, string delimiters)

⇩

"ato z ({} x..........."

Methods :

1.Public Boolean hasMoreTokens()

This method checks whether next token is present or not.If the next token is present then it <u>returns</u> true else false.

1. **Public String nextToken ()**

It is used to read next tokens from the list of tokens.

Example :

```
import java.util.*;
Class Demo
{
Piblic static void main (string args[])
{
String S = "This is an example of string tokens";
StringTokenizer St = new StringTokenizer (s)
while (St.hasMoreTokens())
{
S.O.P (st.nextToken())
}
}
}
```

To install hadoop steps

1) install linux(ubuntu or cent os)

2) install java

3) install eclipse

4) install hadoop

5) set required configurations

6) start all five demons

<u>Steps to install hadoop</u>

1) Download new stable version of hadoop from the apache mirror

   Ex hadoop-1.0.4.tar.gz

2) Extract it under homefolder

3) Set hadoop path and java path in bashrc file

   Ex :

   Open bashrc from the terminal

   ```
   >gedit ~/.bashrc
   ```

   Go to end of bashrc file and set the hadoop path and java path

   ```
   export HADOOP_HOME=home/matuser/hadoop-1.0.4
   export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
   export PATH=$HADOOP_HOME/bin:$JAVA_HOME/bin:$PATH
   ```

   4)set java path in hadoop_env.sh file

   ```
   export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
   ```

   5)set configuration files

   Edit the core-site.xml with the following properties.

   ```
   <configuration>
   <property>
   <name>fs.default.name</name>
   <value>hdfs://localhost:9000</value>
   </property>
   </configuration>
   ```

   <u>fs.default.name:</u> the property fs.default.name is a HDFS filesystem URI,whose is the name node's hostname or IP address and port is the port that the namenode will listen on for RPCs.if no port is specified the default of 8020 is used.

Edit the hdfs-site-xml with the following properties.

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/home/matuser/dfs/data</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>/home/matuser/dfs/name</value>
</property>
</configuration>
```

dfs.replication: it is used to set number of replication factors .in hadoop default replication factor is 3.it can be increased or decreased .in sudo distributed mode replication factor should be 1.

dfs.data.dir: this property specifies a list of the directories for a datanode to store its blocks . A data node round robins writes between its storage directories.

dfs.name,dir: this property specifies a list of the directories where the namenode stores persistent filesystem metadata(the edit log,and filesystem image).A copy of each of the metadata file is stored in each directory for redundancy.

Edit the mapred-site.xml with the following properties

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:9001</value>
</property>
<property>
<name>mapred.local.dir</name>
<value>/home/matuser/mapred/local</value>
```

```
</property>
<property>
<name>mapred.system.dir</name>
<value>/home/matuser/mapred/system</value>
</property>
</configuration>
```

Mapred.job.tracker: specify the hostname or IP address and port that the job tracker will listen on default port is 8021.

Mapred.local.dir: this property specifies the list of directories separated by commas. During mapreduce job ,intermediate data and working files are written to temporary local files.sience this data includes the potentially very large output of map tasks,you need to ensure the mapred.local.dir property which controls the location of local temporary storage.

Mapred.system,dir: mapreduce uses a distributed file system to share files(such as the job JAR file)with the tasktrackers that run the mapreduce tasks
this property is used to specify a directory where these files can be stored.

6) Start five demons(namenode,datanode,secondary namenode,jobtracker,tasktracker)
Note: before starting demons for the first time ,format the namenode.
To format the namenode

```
	>hadoop namenode –format
```

To start five demons run following file (which is present in home/matuser/hadoop-1.0.4/bin)

```
>start-all.sh
```

To check whether five demons are started or not run following command.

```
	>jps
```

It shows demons which are stated as follows.

```
Name Node
Data Node
Secondary Name Node
Job Tracker
Task Tracker
Jps
```

To start only namenode ,data node,secondary name node then run following file

```
>start-dfs.sh
```

To start only job tracker ,task tracker then run following file

```
>start-mapred.sh
```

To stop all five demons then run following file

```
>stop-all.sh
```

To stop only name node ,datanode,SSN then run following files

```
>stop-dfs.sh
```

To stop only job tracker and task tracker then run following file

```
>stop-mapred.sh
```

Hadoop commands

Chmod Command :

It is used to provide permisions to the file or directory

to provide read,write,execute permissions respectively values are 4,2, 1.

4→read

2→write

1→execute

Syntax :

Form the home folder type the following commands

```
Hadoop  fs  -chmod 777  path of the file /directory
```

Ex: Hadoop fs -chmod 777 satya/abc.txt

the above command provides read,write and execute permitions to user ,group of users and other users

Eg: Hadoop fs – chmod 431 satya

The above command provides permissions for

User→only read

Group of users→only write

Others→only execute

To format name node

Syntax : hadoop namenode - format

Stop name node temporarily:-

Syntax:

hadoop dfsadmin - safemode enter.

<u>Start</u>

hadoop dfsadmin - safemode leave.

<u>To check status of name node whether it is safe mode or not:-</u>

hadoop dfsadmin -safemode get.

<u>To check/ to get blocks information.</u>

Hadoop fsck -block.

Fask-file system checkup/health check

Hadoop fsck -report

To create a directory in hadoop

Syntax:

Hadoop fs -mkdir directory name

Ex :

hadoop fs –mkdir firstdir

Ex :

hadoop fs –mkdir seconddir

To copy the file from local hard disk to hadoop

hadoop fs –copyFromLocal source filepath target filepath

Ex:

hadoop fs –copyFromLocal mat.doc firstdir

To copy the file from one hadoop directory to some other hadoop directory

hadoop fs –cp source directory path target directory path

Ex:

hadoop fs –cp firstdir/mat.doc seconddir

To move the file from one hadoop directory to some other hadoop directory

hadoop fs -mv source directory path target directory path

Ex:

hadoop fs –mv firstdir/mat.doc seconddir

To move the file from local file system to hadoop directory

hadoop fs –moveFromLocal source directory path target directory path

Ex:

```
hadoop fs –moveFromLocal mat.doc firstdir
```

to display the contents of any file present in hadoop

```
hadoop fs –cat path of the file
```

Ex:

```
hadoop fs –cat firstdir/mat.doc
```

to display the list of file and directories and file

```
hadoop fs –ls path of the directory
```

Ex:

```
hadoop fs –ls firstdir
```

<u>Steps to create new project using eclipse</u>

1.go to Applications menu→select programing menu item→select eclipse option

Steps to implement Application using Eclipse.

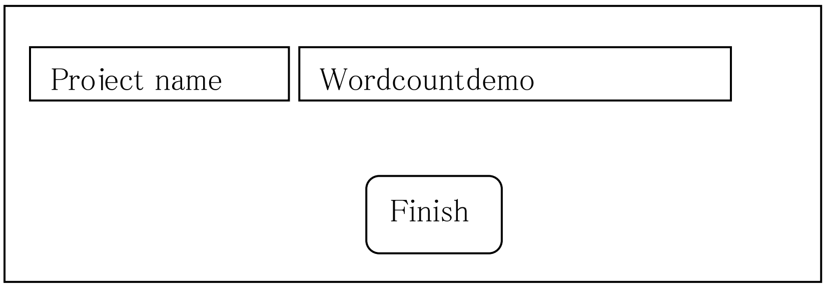
Applications → Programming → Eclipse

Step 1: To Start new application

Go to file menu→New option → Java Project option

then one dialogue box will be opened

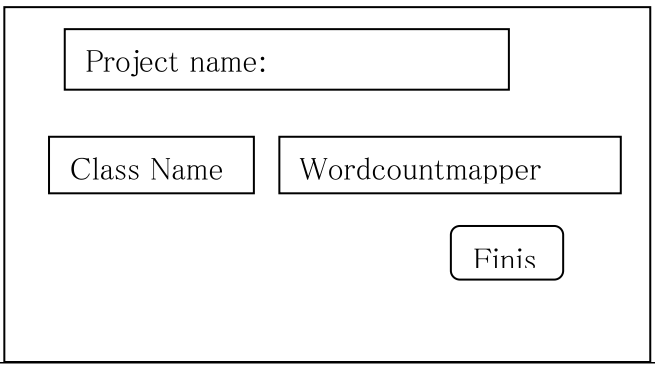
Eg:



Click on Finish button.

Step:2 To add class to project go to package explorer →Right click on Project name→From the pop up menu select new option → Class Option class.

Now it will ask class name



Steps to add Hadoop jar files to application:

Step 1: Go to package Explorer,

↓

Right click on project

↓

↓

Select build path

↓

Configure build path

↓

Then one dialogue box will be opened .

Then click on Add External jars.

select hadoop folder from the dialog box →select all jar files click on open button

Ex:

→Hadoop- ant-20.2-chd305.jar

→ Hadoop-core-cdb30 5.jar etc..

jar → java Archive

After implementing complete project it must be create in jar files.

To create jar files:

Go to package Explorer

↓

Right click on java project

↓

Select Export Option

↓

Select Jar file option

↓

Click on next button

This path can be any existing path followed by jar file name

Ex:

/home/matuser/desktop/anyfilename.jar

$\downarrow$

Click on finish button

http://hadooptraininginhyderabad.co.in   http://hadooptraininginhyderabad.co.in

Hadoop programs:

1. Write a programm to display properties present in configuration files.

http://hadooptraininginhyderabad.co.in/

```
import  java .until.map.entry;
import  org .apache.hadoop.cont.Configuration;
import  org .apache.hadoop.until.ToolRunner;
import  org .apache.hadoop.until.Tool;
import  org .apache.hadoop.cont.Configured;
public class ConfigurationPrinter extends Configured implements Tool
{
private Configuration conf;
static
{
Configuration.addDefaultResource("hdfs-default.xml");
Configuration.addDefaultResource("hdfs-site.xml");
Configuration.addDefaultResource("map-red-default.xml");
Configuration.addDefaultResource("map-red-site.xml");
}
@Override
public Configuration getConf()
{
return conf;
}
@Override
public  void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public  int run(String[]args)throws Exeception
{
```

```
Configuration conf =getConf();
for (entry<String,String>entry:conf)
{
System.out.printf("%s=%s\n",entry.getKey(),entry.getValue());
}
return 0;
}
public static void main(String[]args) throws Exeception
{
int exitcode = ToolRunner.run(new ConfigurationPrinter(),args);
System.exit(exitcode);
}
}
```

Configuration :-

it is a one predefined class which is present in org.apache.hadoop.conf package. It is used to retrieve Configuration class properties & values. Each property is named by a string & type of the value may be one of the several types (such as Boolean,int,long……)

To read properties create object of type configuration class

Ex:-

```
Configuration conf=new Configuration();
Conf.addResource("xml file path")
```

addresource ():is a member function of Configuration class. It takes xml file as a parameter.

Ex:-conf. addResource("core-site.xml")

To get value of a property

Syntax:-

Variable name=conf.get("property name")

Configuration.addDefaultResource("coresite.xml");

it is a static method of Configuration class.

Hadoop comes with a few helper classes for making it easier to run jobs from the command line. GenericOptionsParser is a class that interprets common Hadoop command-line options and sets them on a Configuration object for your application to use as desired. You don't usually use GenericOptionsParser directly, as it's more convenient to implement the Tool interface and run your application with the ToolRunner, which uses GenericOptionsParser internally:

```
public interface Tool extends Configurable {
int run(String [] args) throws Exception;
}
```

Below example shows a very simple implementation of Tool, for running the Hadoop Map Reduce Job.

```
public class WordCountConfigured extends Configured implements Tool {
@Override
public int run(String[] args) throws Exception {
Configuration conf = getConf();

return 0;
}
}
public static void main(String[] args) throws Exception {
int exitCode = ToolRunner.run(new WordCountConfigured(), args);
System.exit(exitCode);
}
```

We make WordCountConfigured a subclass of Configured, which is an implementation of the Configurable interface. All implementations of Tool need to implement Configurable (since Tool extends it), and subclassing Configured is often the easiest way to achieve this. The run() method obtains the Configuration using Configurable's getConf() method, and then iterates over it, printing each property to standard output.

WordCountConfigured's main() method does not invoke its own run() method directly. Instead, we call ToolRunner's static run() method, which takes care of creating a Configuration object for the Tool, before calling its run() method. ToolRunner also uses a GenericOptionsParser to pick up any standard options specified on the command line, and set them on the Configuration instance. We can see the effect of picking up the properties specified in conf/hadoop-localhost.xml by running the following

command:

```
Hadoop WordCountConfigured -conf conf/hadoop-localhost.xml -D
mapred.job.tracker=localhost:10011 -D mapred.reduce.tasks=n
```

Options specified with -D take priority over properties from the configuration files. This is very useful: you can put defaults into configuration files, and then override them with the -D option as needed. A common example of this is setting the number of reducers for a MapReduce job via -D mapred.reduce.tasks=n. This will override the number of reducers set on the cluster, or if set in any client-side configuration files. The other options that GenericOptionsParser and ToolRunner support are listed in Table.

Configurable:-

It is a one predefined interface. It contains following abstract methods.

1.Configuration getConf()

It is used to get configurations object

2. Configuration setConf(Configuration conf)

It is used to set Configuration

Configured:-

It is a one predefined class which is derived from Configurable interface.Abstract methods of Configurable are already implemented in Configured class.(getConf() & setConf() are already implemented in configured class).

Tool:-

It is a one predefined interface. It is the sub interface of Configurable interface.

Syntax:-

```
interface Tool extends Configurable
{
int run(String[]args);                //abstract method.
}
```

Note :the class which impliments Tool interface must override 3 abstract methods.(getConf(),setConf(),run()).

in the above example ConfigurationPrinter class is implementing Tool interface.So it must override all the 3 abstract methods.otherwise ConfigurationPrinter class becomes abstract class.

ToolRunner:- it is a predefined class which contains following static methods

run() is over loaded method of ToolRunner

static int run(Tool tool,String[]args);

It should be the sub class object of tool interface.

Syntax:-

static int run(Configuration conf, Tool tool,String[]args);

The above run method of ToolRunner class first will call setConf() to set the configuration and next it will make a call to run method of Tool interface to run the job.

MAP: -

It is a predefined interface which is present in until package.

Entry is a inner interface of map which contains following methods.

Syntax:

Object getKey();

Object getValue();

JAR:- after implementing the entire program we have to create a jar file.

Steps to create a jar file:-

1.goto package explorer and right on project name and select export option.

2.select java-jar file and click on next button.

3.jar file:path of the file browse

Click on finish.

4.To run the application or a program goto terminal

```
$ hadoop jar <jar file name>  <main class name>  <input file path> <output file path>
```

To execute the above example i/p & o/p file paths are not required. So the command is

```
$ hadoop jar  /home/matuser/documents/configurationdemo.jar ConfigurationPrinter.
```

SampleMapper.java

```
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Mapper;
public class SampleMapper extends Mapper<LongWritable,Text,Text,LongWritable>
{
String msg;
public void setup(Context context)throws IOException,InterruptedException
{
msg="this is setup method of Mapper\n";
}
@Override
protected void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException
{
msg=msg+"map method is called for  "+value.toString()+"\n";


}
protected void cleanup(Context context)throws IOException,InterruptedException
{
msg=msg+"this is cleaup method of mapper\n";
context.write(new Text(msg),new LongWritable(msg.length()));
}
}
```

SampleReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class SampleReducer extends Reducer<Text, LongWritable, Text, NullWritable>
{
String msg;
public void setup(Context context)throws IOException,InterruptedException
{
msg="this is setup method of Reducer\n";

}
@Override
protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws
IOException,InterruptedException
{
msg=key.toString()+msg+"this is reducer method\n";

}
protected void cleanup(Context context)throws IOException,InterruptedException
{           msg=msg+"this is clean up method of reducer\n";
context.write(new Text(msg),NullWritable.get());

}
}
```

Samplejob.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class SampleJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{

Job samplejob=new Job(getConf());
samplejob.setJobName("mat word count");
samplejob.setJarByClass(this.getClass());
samplejob.setMapperClass(SampleMapper.class);
samplejob.setReducerClass(SampleReducer.class);
samplejob.setMapOutputKeyClass(Text.class);
samplejob.setMapOutputValueClass(LongWritable.class);
samplejob.setOutputKeyClass(Text.class);
samplejob.setOutputValueClass(NullWritable.class);
FileInputFormat.setInputPaths(samplejob,new Path(args[0]));
FileOutputFormat.setOutputPath(samplejob,new Path(args[1]));
return samplejob.waitForCompletion(true)==true? 0:1;
}
public static void main(String []args)throws Exception
{
ToolRunner.run(new Configuration(),new SampleJob(),args);
}

}
```

http://hadooptraininginhyderabad.co.in

WordcountMapper.java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.Text;
//import org.apache.hadoop.mapreduce.Counter;
public class WordCountMapper extends Mapper<LongWritable,Text,Text,LongWritable>
{
private Text temp=new Text();
private final static LongWritable one=new LongWritable(1);
@Override
protected void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException
{
String str=value.toString();
StringTokenizer strtock=new StringTokenizer(str);
while(strtock.hasMoreTokens())
temp.set(strtock.nextToken());
context.write(temp,one);
}

}
```

WordCountReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;

public class WordCountReducer extends Reducer<Text,LongWritable,Text,LongWritable>
{
@Override
protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws
IOException,InterruptedException
{
long sum=0;
while(value.iterator().hasNext())
{
sum+=value.iterator().next().get();
}
context.write(key,new LongWritable(sum));
}

}
```

Wordcountcombiner.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;

public class WordCountCombiner extends Reducer<Text,LongWritable,Text,LongWritable>
{
@Override
protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws
IOException,InterruptedException
{
long sum=0;
while(value.iterator().hasNext())
{
sum+=value.iterator().next().get();
}
context.write(key,new LongWritable(sum));
}

}
```

WordcountPartioner.java

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
public class WordCountPartitioner extends Partitioner<Text,LongWritable>
{
@Override
public int getPartition(Text key,LongWritable value,int noOfReducers)
{
String tempString=key.toString();
return (tempString.toLowerCase().charAt(0)-'a')%noOfReducers;
}
}
```

Wordcount Job

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
//import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
//import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class WordCountJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{

Job wordcountjob=new Job(getConf());
wordcountjob.setJobName("mat word count");
wordcountjob.setJarByClass(this.getClass());
wordcountjob.setCombinerClass(WordCountCombiner.class);
wordcountjob.setMapperClass(WordCountMapper.class);
wordcountjob.setReducerClass(WordCountReducer.class);
wordcountjob.setNumReduceTasks(26);
wordcountjob.setMapOutputKeyClass(Text.class);
wordcountjob.setMapOutputValueClass(LongWritable.class);
wordcountjob.setOutputKeyClass(Text.class);
wordcountjob.setOutputValueClass(LongWritable.class);
wordcountjob.setPartitionerClass(WordCountPartitioner.class);
//wordcountjob.setInputFormatClass(TextInputFormat.class);
//wordcountjob.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.setInputPaths(wordcountjob,new Path(args[0]));
FileOutputFormat.setOutputPath(wordcountjob,new Path(args[1]));
return wordcountjob.waitForCompletion(true)==true? 0:1;
}
public static void main(String []args)throws Exception
{
ToolRunner.run(new Configuration(),new WordCountJob(),args);
}

}
```

How map reduce program works:

Map reduce works by breaking the processing into two phases

1) Map Phase

2) Reduce Phase

Each phase has key value pairs as i/p & o/p.Input key value pair of map phase is dermied by the type of i/p for matter being used,where as rest of the key value pair types may be choosen by the programmer.The programmer can write required logic by overriding map() reduce() functions of Mapper,Reducer abstract classes. Mapper is generic type with four formal parameters which specifies input key,input value,o/p key,and o/p value types of map function.The Reducer is generic class with four formal type of parameters that specify the input key,input value,o/p key and o/p value of reducer function.

Reducer aggregates the map o/p so one needs to remember the map o/p key type,Map o/p value type should match with reucer i/p key type and reducer i/p value type.Hadoop uses its own data types rather than using java data types.These data types are optimized for network serialization.These data types are found in org.apache.hadoop.io package.here LongWritable and Text corresponding to long and String respectively.For above eg: Input key is line offset code in the file & values is the entire line.

The third parameter is Context object which allows the i/p and o/p from the task.It is only supplies to the Mapper and Reducer.

In above eg: take the value which is entire line & convert it into string using **tostring()** method in Text class.Split the line into words using StringTokenizer.Iterate over all the words in the line & o/p word and its count 1.context has write() to write map o/p.Map o/p are stored in Local File Sysem where mapper has run.Also map o/p are stored in sorted order,sorted on key.

Note:If the job is map only job,map o/p are not stored and directly stored in the configured file system in core-site.xml in most cases it is HFDS.

Reducer program:

The WordCountReducer extends reducer
<key in,value in,key out,value out> with specific types like Text,Int writable,Text Intwritable.
Note: The word count reducer i/p key type is Text & value type is int writable,match with o/p key type & value type of WordCount Mapper.

Reucer first copies all the map o/p and merges them.This process is called suffle &

sorting. During this phases all values corresponding to each key will be aggregated. Reducer method is called for each key with all the values aggregated sorted in the reducer method,literate over all the values and finally counting how many of them.Finally writing the reduce method o/p which is word and its count using write method provided by context class.Finally we need to provide a driver class ,which submits the job to the hadoop clusters with our own Mapper & Reducer implementation.

Driver program:

Created a job object with **Configuration** object and name of the job being the arguments.

Configuration is an object through which hadoop reads all the configuration properties mentioned in **coresite.xml,Hdfs-site.xml and mapped-site.xml** by passing conf object to the job object it retrives the data from all configuration files during its life cycle. User can define their own property with a value(value types could be any type)using setter methods provided in the **Configuration** object like

conf.setInt(value type)

Hence the configuration is very useful if you need to pass a small piece of meta data to your tasks.

To retrive the values from the task using **Context** object & get configuration object using **context.getConfiguration()** then get the required metadata.

Job object has been specified with all classes(**Mapper,Reducer**.....)

Input path is mentioned using the static method of **FileInputFormatter** class's **addInputPath()** having job instance and input directory as argument.Similarly o/p directory has been mentioned using static method provided by **FileOutputFormatter** class . Finally submit the job to cluster

using **waitForCompletion()** method wait for jobs completion.Once job completes fully,find the o/p of the job in the o/p directory.

A Job object forms the specifications of the job and gives you control over how the job is run.when we run this job on a hadoop cluster,we wil package the code in to JAR file (which hadoop will distribute around the cluster)Rather then explicitly specify the name of the JAR file. We can pass a class in the jobs **setJarByClass()** method which hadoop will use to locate the relevant JAR files by looking for JAR file containing this class.

Input path is specified by calling the static **addInputPath()** on **FileInputFormater** and it can be a single file,a directory (in this case it takes all file from the dir forms input, for all the files)or a file pattern (like *.text or *.doc).As the name suggests **addInputPath()** can be called more than once to use input from multiple paths.The o/p path is specified by static **setOutputPath()** on **FileOutputFormatter**.It specifies where exactly o/p file must be written by the reducer.The directory should not exist before running the job because Hadoop will complain and not run the job.This precaution is to prevent data loss (it can be over written by o/p of a long job with another). We can specify the Map and reducer types to use the **setMapperClass()** and **setReducerClass()** methods.

The **setOutputkeyClass()** and **setOutputValueClass()** method control the o/p types for the map() and reduce() functions,which are often the same.If they are different the map o/p types can be set using the methods **setMapOutputKeyClass()** and **setMapOutputValueKeyClass()** Then we are ready to run the job.The waitForCompletiton() method on job submits the job & wait for it to finish.

The return value of the **waitForCompletion()** is a Boolean indicating success(true) or failure(false) which we translate in to the program exit code 0 or 1.

Job class

setMapOutputKeyClass

public void setMapOutputKeyClass(Class<?> theClass)throws IllegalStateException

Set the key class for the map output data. This allows the user to specify the map output key class to be different than the final output value class.

Parameters:

theClass - the map output key class.

Throws:

IllegalStateException - if the job is submitted

setMapOutputValueClass

public void setMapOutputValueClass(Class<?> theClass)throws IllegalStateException

Set the value class for the map output data. This allows the user to specify the map output value class to be different than the final output value class.

Parameters:TheClass - the map output value class.

Throws: IllegalStateException - if the job is submitted

setOutputKeyClass

public void setOutputKeyClass(Class<?> theClass)throws IllegalStateException

Set the key class for the job output data.

Parameters:

theClass - the key class for the job output data.

Throws:

IllegalStateException - if the job is submitted

setOutputValueClass

public void setOutputValueClass(Class<?> theClass) throws IllegalStateException

Set the value class for job outputs.

Parameters: theClass - the value class for job outputs.

Throws: IllegalStateException - if the job is submitted

setJobName

public void setJobName(String name)throws IllegalStateException

Set the user-specified job name.

Parameters: name - the job's new name.

Throws:IllegalStateException - if the job is submitted

setSpeculativeExecution

public void setSpeculativeExecution(boolean speculativeExecution)

Turn speculative execution on or off for this job.

Parameters: speculativeExecution - true if speculative execution should be turned on, else false.

isComplete

public boolean isComplete()throws IOException

Check if the job is finished or not. This is a non-blocking call.

Returns: true if the job is complete, else false.

Throws: IOException

isSuccessful

public boolean isSuccessful() throws IOException

Check if the job completed successfully.

Returns:true if the job succeeded, else false.

Throws: IOException

killJob

public void killJob() throws IOException

Kill the running job. Blocks until all job tasks have been killed as well. If the job is no longer running, it simply returns.

Throws: IOException

killTask

public void killTask(TaskAttemptID taskId)throws IOException

Kill indicated task attempt.

Parameters:

taskId - the id of the task to be terminated.

Throws:IOException

submit

public void submit() throws IOException,InterruptedException,ClassNotFoundException

Submit the job to the cluster and return immediately.

Throws: IOException ,InterruptedException ,ClassNotFoundException,waitForCompletion

public boolean waitForCompletion(boolean verbose)throws IOException,InterruptedException, ClassNotFoundException

Submit the job to the cluster and wait for it to finish.

Parameters:

verbose - print the progress to the user

Returns:true if the job succeeded

Throws: IOException - thrown if the communication with the JobTracker is lost

InterruptedException ,ClassNotFoundException

setInputFormatClass

public void setInputFormatClass(Class<? extends InputFormat> cls)throws IllegalStateException

Set the InputFormat for the job.

Parameters: cls - the InputFormat to use

Throws:IllegalStateException - if the job is submitted

setOutputFormatClass

public void setOutputFormatClass(Class<? extends OutputFormat> cls)throws IllegalStateException

Set the OutputFormat for the job.

Parameters:

cls - the OutputFormat to use

Throws:

IllegalStateException - if the job is submitted

setMapperClass

public void setMapperClass(Class<? extends Mapper> cls)

throws IllegalStateException

Set the Mapper for the job.

Parameters:cls - the Mapper to use

Throws: IllegalStateException - if the job is submitted

setJarByClass

public void setJarByClass(Class<?> cls)

Set the Jar by finding where a given class came from.

Parameters: cls - the example class

getJar

public String getJar()

Get the pathname of the job's jar.

Overrides: getJar in class JobContext

Returns:the pathname

setCombinerClass

public void setCombinerClass(Class<? extends Reducer> cls) throws IllegalStateException

Set the combiner class for the job.

Parameters:cls - the combiner to use

Throws: IllegalStateException - if the job is submitted

setReducerClass

public void setReducerClass(Class<? extends Reducer> cls)throws IllegalStateException

Set the Reducer for the job.

Parameters: cls - the Reducer to use

Throws:IllegalStateException - if the job is submitted

setPartitionerClass

public void setPartitionerClass(Class<? extends Partitioner> cls)

throws IllegalStateException

Set the Partitioner for the job.

Parameters: cls - the Partitioner to use

Throws: IllegalStateException - if the job is submitted

setMapOutputKeyClass

setNumReduceTasks

public void setNumReduceTasks(int tasks)throws IllegalStateException

Set the number of reduce tasks for the job.

Parameters: tasks - the number of reduce tasks

Throws:IllegalStateException - if the job is submitted

Combiner

Combiner are used to reduce the the amount of the data being transferred over the network It is used to minimize data to be transferred from mapper to reducer.It uses the optimum usages of the network bandwidth . hadoop allows the user to specify a combiner function to run on mapper output.The o/p of mapper becomes i/p of reducer.Combiners are treated as local reducer they run by consuming mapper o/p and run on the same machine,where mapper has run earlier.Hadoop does not provide guarantee on combiner function execution.Hadoop frame work may call combiner function zero or more times for a particular mapper o/p.

Let us imagine the first mapper o/p is

```
Hadoop,1
Hadoop,1
     .
     .
10 times

Mat,1
Mat,1
   .
   .
20 times
```

Second mapper o/p is

```
    Hadoop,1
          .
          .
          .
   Hadoop,1
  20 times
       Is,1
         .
         .
       Is,1
  20 times
```

→We can use combiner function like reducer function,we can count the word frequency of above o/p.

First combiner o/p

```
Hadoop,10
  Mat,20
```

Second combiner o/p

```
Hadoop,20
      is,20
```

http://hadooptraininginhyderabad.co.in/

It will give to reducer as i/p.Combiner function does not replace reducer() function, combiner is also implemented by extending Reducer abstract class & overriding reduce() method.

Partitioner

A partitioner in mapreduce partitiones the key space.the partitioner is used to derive the Partition to which key value pair belongs.partitioner partitioning the keys of the intermediate Map-outputs

Number of Partitions=number of reduce tasks

Partitioners run on the same machine where the mapper has finished his execution earlier.

Entire mapper o/p is sent to partitioner and partitioner forms(number of reducer tasks) groups from the mapper o/p.

By default hadoop frame work is hash based partitioner.This partitioner partitions the keyspace by using hash code.The following is logic for hash partioner to determine reducer for particular key.

```
return (key.hashcode & integer Max-value)% no of reducers
```

We can customize partion logic hadoop provides.Partioner abstract class with a single method which can be extended to write custom partioner

```
public abstract class Partitioner <key,value>
  {
        public abstract int getPartition(key,value, no of reducer);
}
```

getPartition() returns partition number for a given key

In word count example requirement is all the words which starts with ‘a’ should go to one reducer & all the words which starts with ‘B’ should go to another reducer & so on..

In the case no of reducers are ‘26’.

FormatDataMapper.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class SampleJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{

Job samplejob=new Job(getConf());
samplejob.setJobName("mat word count");
samplejob.setJarByClass(this.getClass());
samplejob.setMapperClass(SampleMapper.class);
samplejob.setReducerClass(SampleReducer.class);
samplejob.setMapOutputKeyClass(Text.class);
samplejob.setMapOutputValueClass(LongWritable.class);
samplejob.setOutputKeyClass(Text.class);
samplejob.setOutputValueClass(NullWritable.class);
FileInputFormat.setInputPaths(samplejob,new Path(args[0]));
FileOutputFormat.setOutputPath(samplejob,new Path(args[1]));
return samplejob.waitForCompletion(true)==true? 0:1;
}
public static void main(String []args)throws Exception
{
ToolRunner.run(new Configuration(),new SampleJob(),args);
}

}
```

FormatDataReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class FormatDataReducer extends Reducer<LongWritable, Text, LongWritable, Text>{
Long xadd=(long) 01;
public void reduce(LongWritable key, Iterable<Text> values, Context context) throws IOException,
InterruptedException{
LongWritable sKey=new LongWritable(0);
Text txtr=new Text();
for (Text val:values) {
String mStr=val.toString();
sKey.set(xadd++);
txtr.set(mStr);
context.write(sKey, txtr);
}
}
}
```

FormatJob.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class FormatJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{

Job formatjob=new Job(getConf());
formatjob.setJobName("mat sed count");
formatjob.setJarByClass(this.getClass());
formatjob.setMapperClass(FormatDataMapper.class);
formatjob.setOutputKeyClass(LongWritable.class);
formatjob.setOutputValueClass(Text.class);
formatjob.setInputFormatClass(TextInputFormat.class);
formatjob.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.setInputPaths(formatjob,new Path(args[0]));
FileOutputFormat.setOutputPath(formatjob,new Path(args[1]));
return formatjob.waitForCompletion(true)==true? 0:-1;
}
public static void main(String []args)throws Exception
{
Configuration conf1=new Configuration();
conf1.set("Batch_Id",args[2]);
conf1.set("Run_Id",args[3]);
ToolRunner.run(conf1,new FormatJob(),args);
}
}
```

MaxLengthWordJob.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
```

```
public class MaxLengthWordJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{
Job maxlengthwordjob=new Job(getConf());
maxlengthwordjob.setJobName("mat maxlength word");
maxlengthwordjob.setJarByClass(this.getClass());
maxlengthwordjob.setMapperClass(MaxLengthWordMapper.class);
maxlengthwordjob.setReducerClass(MaxLengthWordReducer.class);
maxlengthwordjob.setMapOutputKeyClass(Text.class);
maxlengthwordjob.setMapOutputValueClass(LongWritable.class);
maxlengthwordjob.setOutputKeyClass(Text.class);
maxlengthwordjob.setOutputValueClass(LongWritable.class);
FileInputFormat.setInputPaths(maxlengthwordjob,new Path(args[0]));
FileOutputFormat.setOutputPath(maxlengthwordjob,new Path(args[1]));
return maxlengthwordjob.waitForCompletion(true)==true? 0:1;
}
public static void main(String []args)throws Exception
{
ToolRunner.run(new Configuration(),new MaxLengthWordJob(),args);
}

}
```

MaxLengthWordMapper.java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.Text;
public class MaxLengthWordMapper extends Mapper<LongWritable,Text,Text,LongWritable>
{
String maxWord;
public void setup(Context context)throws IOException,InterruptedException
{
maxWord=new String();
}
@Override
protected void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException
{
String nextToken;
StringTokenizer st=new StringTokenizer(value.toString());
while(st.hasMoreTokens())
{
nextToken=st.nextToken();
if(nextToken.length()>maxWord.length())
{
```

```
maxWord=nextToken;
}

}
}
protected void cleanup(Context context)throws IOException,InterruptedException
{
context.write(new Text(maxWord), new LongWritable(maxWord.length()));
}

}
```

MaxLengthWordReducer.java

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.io.Text;
public class MaxLengthWordReducer extends Reducer<Text,LongWritable,Text,NullWritable>
{
String maxWord;
protected void setup(Context context)throws IOException,InterruptedException
{
maxWord=new String();
}
@Override
protected void reduce(Text key,Iterable<LongWritable> value,Context context)throws
IOException,InterruptedException
{
if(key.toString().length()>maxWord.length())
{
maxWord=key.toString();
}
}
protected void cleanup(Context context)throws IOException,InterruptedException
{
context.write(new Text(maxWord),NullWritable.get());
}
}
```

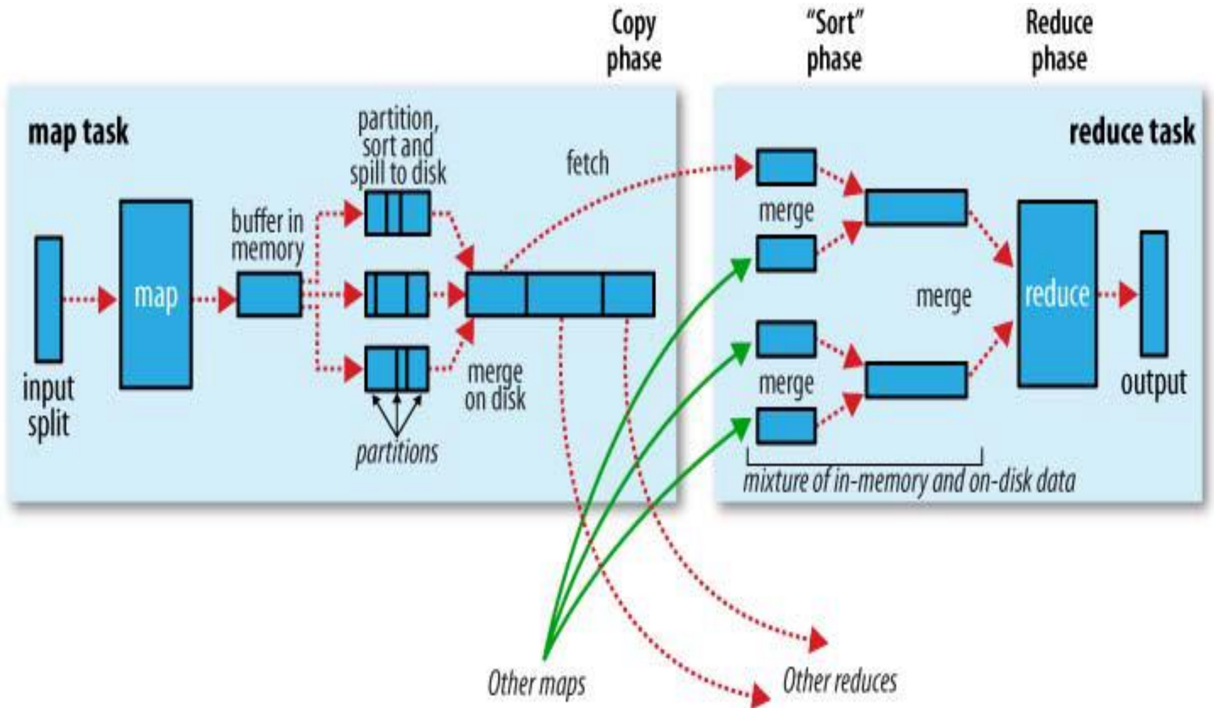
Primejob.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.util.Tool;
public class PrimeJob implements Tool{
private Configuration conf;
@Override
public Configuration getConf()
{
return conf;
}
@Override
public void setConf(Configuration conf)
{
this.conf=conf;
}
@Override
public int run(String []args)throws Exception
{

Job primejob=new Job(getConf());
primejob.setJobName("mat prime Numbers");
primejob.setJarByClass(this.getClass());
primejob.setMapperClass(PrimeMapperDemo.class);
primejob.setMapOutputKeyClass(Text.class);
primejob.setMapOutputValueClass(LongWritable.class);
primejob.setNumReduceTasks(0);
primejob.setOutputKeyClass(Text.class);
primejob.setOutputValueClass(LongWritable.class);
FileInputFormat.setInputPaths(primejob,new Path(args[0]));
FileOutputFormat.setOutputPath(primejob,new Path(args[1]));
return primejob.waitForCompletion(true)==true? 0:1;
}
public static void main(String []args)throws Exception
{
ToolRunner.run(new Configuration(),new PrimeJob(),args);
}

}
```

PrimeMapperDemo.java

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;
public class PrimeMapperDemo extends Mapper<LongWritable,Text,Text,NullWritable>
{
@Override
protected void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException
{
long n,f=0;
String str=value.toString();
String val;
StringTokenizer st=new StringTokenizer(str);
while(st.hasMoreTokens())
{
f=0;
val=st.nextToken();
n=Long.parseLong(val);
if(n==1)
continue;
for(int i=2;i<=n/2;i++)
{
if(n%i==0)
{
```

```
f=1;
break;
}
}
if(f==0)
context.write(new Text(val),NullWritable.get());
}
}
}
```



The Map Side:

When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons. *Shuffle and sort in MapReduce* Each map task has a circular memory buffer that it writes the output to. The buffer is 100 MB by default, a size which can be tuned by changing the io.sort.mb property. When the contents of the buffer reaches a certain threshold size (io.sort.spill.per cent, default 0.80, or 80%) a background thread will start to *spill* the contents to disk. Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete. † The term *shuffle* is actually imprecise, since in some contexts it refers to only the part of the process where map outputs are fetched by reduce tasks. In this section, we take it to mean the whole process from the point where a map produces output to where a reduce consumes input. Shuffle and Sort Spills are written in round-robin fashion to the directories specified by the mapred.local.dir property, in a job-specific subdirectory. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be

sent to. Within each partition, the background thread performs an in-memory sort by key, and if there is a combiner function, it is run on the output of the sort. Each time the memory buffer reaches the spill threshold, a new spill file is created, so after the map task has written its last output record there could be several spill files. Before the task is finished, the spill files are merged into a single partitioned and sorted output file. The configuration property io.sort.factor controls the maximum number of streams to merge at once; the default is 10. If a combiner function has been specified, and the number of spills is at least three (the value of the min.num.spills.for.combine property), then the combiner is run before the output file is written. Recall that combiners may be run repeatedly over the input without affecting the final result. The point is that running combiners makes for a more compact map output, so there is less data to write to local disk and to transfer to the
reducer.

It is often a good idea to compress the map output as it is written to disk, since doing so makes it faster to write to disk, saves disk space, and reduces the amount of data to transfer to the reducer. By default the output is not compressed, but it is easy to enable by setting mapred.compress.map.output to true. The compression library to use is specified by mapred.map.output.compression.codec; see "Compression" on page 77 for more on compression formats. The output file's partitions are made available to the reducers over HTTP. The number of worker threads used to serve the file partitions is controlled by the task tracker.http.threads property—this setting is per tasktracker, not per map task slot. The default of 40 may need increasing for large clusters running large jobs. The Reduce Side Let's turn now to the reduce part of the process. The map output file is sitting on the local disk of the tasktracker that ran the map task (note that although map outputs always get written to the local disk of the map tasktracker, reduce outputs may not be), but now it is needed by the tasktracker that is about to run the reduce task for the partition. Furthermore, the reduce task needs the map output for its particular partition
from several map tasks across the cluster. The map tasks may finish at different times, so the reduce task starts copying their outputs as soon as each completes. This is known as the *copy phase* of the reduce task.

The reduce task has a small number of copier threads so that it can fetch map outputs in parallel. The default is five threads, but this number can be changed by setting the mapred.reduce.parallel.copies property.

How MapReduce Works How do reducers know which tasktrackers to fetch map output from? As map tasks complete successfully, they notify their parent tasktracker of the status update, which in turn notifies the jobtracker. These notifications are transmitted over the heartbeat communication mechanism described earlier. Therefore, for a given job, the jobtracker knows the mapping between map outputs and tasktrackers. A thread in the reducer periodically asks the jobtracker for map output locations until it has retrieved them all. Tasktrackers do not delete map outputs from disk as soon as the first reducer has retrieved them, as the reducer may fail. Instead, they wait until they are told to delete them by the jobtracker, which is after the job has completed.

The map outputs are copied to the reduce tasktracker's memory if they are small enough (the buffer's size is controlled by mapred.job.shuffle.input.buffer.percent, which specifies the proportion of the heap to use for this purpose); otherwise, they are copied to disk. When the in-memory buffer reaches a threshold size (controlled by mapred.job.shuffle.merge.percent), or reaches a threshold number of map outputs (mapred.inmem.merge.threshold), it is merged and spilled to disk. As the copies accumulate on disk, a background thread merges them into larger, sorted files.

This saves some time merging later on. Note that any map outputs that were compressed (by the map task) have to be decompressed in memory in order to perform a merge on them. When all the map outputs have been copied, the reduce task moves into the *sort phase* (which should properly be called the *merge* phase, as the sorting was carried out on the map side), which merges the map outputs, maintaining their sort ordering. This is done in rounds. For example, if there were 50 map outputs, and the *merge factor* was 10 (the default, controlled by the io.sort.factor property, just like in the map's merge), then there would be 5 rounds. Each round would merge 10 files into one, so at the end there would be five intermediate files.

Rather than have a final round that merges these five files into a single sorted file, the merge saves a trip to disk by directly feeding the reduce function in what is the last phase: the *reduce phase*. This final merge can come from a mixture of in-memory and on-disk segments. Shuffle and Sort The number of files merged in each round is actually more subtle than this example suggests. The goal is to merge the minimum number of files to get to the merge factor for the final round. So if there were 40 files, the merge would not merge 10 files in each of the four rounds to get 4 files. Instead, the first round would merge only 4 files, and the subsequent three rounds would merge the full 10 files. The 4 merged files, and the 6 (as yet unmerged) files make a total of 10 files for the final round.

Note that this does not change the number of rounds, it’s just an optimization to minimize the amount of data that is written to disk, since the final round always merges directly into the reduce. During the reduce phase the reduce function is invoked for each key in the sorted output. The output of this phase is written directly to the output filesystem, typically HDFS. In the case of HDFS, since the tasktracker node is also running a datanode, the first block replica will be written to the local disk.

Fair Schedular:

Purpose

This document describes the Fair Scheduler, a pluggable MapReduce scheduler that provides a way to share large clusters.

Introduction

Fair scheduling is a method of assigning resources to jobs such that all jobs get, on average, an equal share of resources over time. When there is a single job running, that job uses the entire cluster. When other jobs are submitted, tasks slots that free up are assigned to the new jobs, so that each job gets roughly the same amount of CPU time. Unlike the default Hadoop scheduler, which forms a queue of jobs, this lets short jobs finish in reasonable time while not starving long jobs. It is also an easy way to share a cluster between multiple of users. Fair sharing can also work with job priorities - the priorities are used as weights to determine the fraction of total compute time that each job gets.

The fair scheduler organizes jobs into *pools*, and divides resources fairly between these pools. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster. It is also possible to set a job's pool based on the user's Unix group or any jobconf property. Within each pool, jobs can be scheduled using either fair sharing or first-in-first-out (FIFO) scheduling.

In addition to providing fair sharing, the Fair Scheduler allows assigning guaranteed *minimum shares* to pools, which is useful for ensuring that certain users, groups or production applications always get sufficient resources. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split between other pools.

If a pool's minimum share is not met for some period of time, the scheduler optionally supports *preemption* of jobs in other pools. The pool will be allowed to kill tasks from other pools to make room to run. Preemption can be used to guarantee that "production" jobs are not starved while also allowing the Hadoop cluster to also be used for experimental and research jobs. In addition, a pool can also be allowed to preempt tasks if it is below half of its fair share for a configurable timeout (generally set larger than the minimum share preemption timeout). When choosing tasks to kill, the fair scheduler picks the most-recently-launched tasks from over-allocated jobs, to minimize wasted computation. Preemption does not cause the preempted jobs to fail, because Hadoop jobs tolerate losing tasks; it only makes them take longer to finish.

The Fair Scheduler can limit the number of concurrent running jobs per user and per pool. This can be useful when a user must submit hundreds of jobs at once, or for ensuring that intermediate data does not fill up disk space on a cluster when too many concurrent jobs are running. Setting job limits causes jobs submitted beyond the limit to wait until some of the user/pool's earlier jobs finish. Jobs to run from each user/pool are chosen in order of priority and then submit time.

Finally, the Fair Scheduler can limit the number of concurrent running tasks per pool. This can be useful when jobs have a dependency on an external service like a database or web service that could be overloaded if too many map or reduce tasks are run at once.

Installation

To run the fair scheduler in your Hadoop installation, you need to put it on the CLASSPATH. The easiest way is to copy the *hadoop-*-fairscheduler.jar* from*HADOOP_HOME/build/contrib/fairscheduler* to *HADOOP_HOME/lib*. Alternatively you can modify *HADOOP_CLASSPATH* to include this jar, in*HADOOP_CONF_DIR/hadoop-env.sh*

You will also need to set the following property in the Hadoop config file *HADOOP_CONF_DIR/mapred-site.xml* to have Hadoop use the fair scheduler:

```
<property>
<name>mapred.jobtracker.taskScheduler</name>
<value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

Once you restart the cluster, you can check that the fair scheduler is running by going to *http://<jobtracker URL>/scheduler* on the JobTracker's web UI. A "job scheduler administration" page should be visible there. This page is described in the Administration section.

If you wish to compile the fair scheduler from source, run *ant package* in your HADOOP_HOME directory. This will build *build/contrib/fair-scheduler/hadoop-*-fairscheduler.jar*.

Configuration

The Fair Scheduler contains configuration in two places -- algorithm parameters are set in *HADOOP_CONF_DIR/mapred-site.xml*, while a separate XML file called the *allocation file*, located by default in *HADOOP_CONF_DIR/fair-scheduler.xml*, is used to configure pools, minimum shares, running job limits and preemption timeouts. The allocation file is reloaded periodically at runtime, allowing you to change pool settings without restarting your Hadoop cluster.

For a minimal installation, to just get equal sharing between users, you will not need to edit the allocation file.

Scheduler Parameters in mapred-site.xml

The following parameters can be set in *mapred-site.xml* to affect the behavior of the fair scheduler:

Basic Parameters

Name	Description
mapred.fairscheduler.preemption	Boolean property for enabling preemption. Default: false.
mapred.fairscheduler.pool	Specify the pool that a job belongs in. If this is specified then mapred.fairscheduler.p
mapred.fairscheduler.poolnameproperty	Specify which jobconf property is used to determine the pool that a job belongs in. St each user). Another useful value is *mapred.job.queue.name* to use MapReduce's "queue below). mapred.fairscheduler.poolnameproperty is used only for jobs in which mapred.f
mapred.fairscheduler.allow.undeclared.pools	Boolean property for enabling job submission to pools not declared in the allocation fil
mapred.fairscheduler.allocation.file	Can be used to have the scheduler use a different allocation file than the default one *scheduler.xml*). Must be an absolute path to the allocation file.

Advanced Parameters

Name	Description
mapred.fairscheduler.sizebasedweight	Take into account job sizes in calculating their weights for fair sharing. By default, weights are only based on will make them based on the size of the job (number of tasks needed) as well,though not linearly (the weight w number of tasks needed). This lets larger jobs get larger fair shares while still providing enough of a share to Boolean value, default: false.
mapred.fairscheduler.preemption.only.log	This flag will cause the scheduler to run through the preemption calculations but simply log when it wishes to preempting the task. Boolean property, default: false. This property can be useful for doing a "dry run" of pre that you have not set timeouts too aggressively. You will see preemption log messages in your JobTracker's ou *jobtracker-*.log*). The messages look as follows: Should preempt 2 tasks for job_20090101337_0001: tasksDueToMinShare = 2, tasksDueToFairShare = 0

mapred.fairscheduler.update.interval	Interval at which to update fair share calculations. The default of 500ms works well for clusters with fewer than 500 node JobTracker for larger clusters. Integer value in milliseconds, default: 500.
mapred.fairscheduler.preemption.interval	Interval at which to check for tasks to preempt. The default of 15s works well for timeouts on the order of minutes. It is n than this amount, but you can use this value to make preemption computations run more often if you do set such timeouts. A small, however, as it becomes less than the inter-heartbeat interval. Integer value in milliseconds, default: 15000.
mapred.fairscheduler.weightadjuster	An extension point that lets you specify a class to adjust the weights of running jobs. This class should implement the *Weig* example implementation - *NewJobWeightBooster*, which increases the weight of jobs for the first 5 minutes of their lifeti set the weightadjuster property to the full class name, org.apache.hadoop.mapred.NewJobWeightBooster. NewJobWeightB setting the duration and boost factor. *mapred.newjobweightbooster.factor* Factor by which new jobs weight should be boosted. Default is 3. *mapred.newjobweightbooster.duration* Boost duration in milliseconds. Default is 300000 for 5 minutes.
mapred.fairscheduler.loadmanager	An extension point that lets you specify a class that determines how many maps and reduces can run on a given TaskTracker LoadManager interface. By default the task caps in the Hadoop config file are used, but this option could be used to make t utilization for example.
mapred.fairscheduler.taskselector	An extension point that lets you specify a class that determines which task from within a job to launch on a given tracker. T policy (e.g. keep some jobs within a particular rack) or the speculative execution algorithm (select when to launch speculativ Hadoop's default algorithms from JobInProgress.
mapred.fairscheduler.eventlog.enabled	Enable a detailed log of fair scheduler events, useful for debugging. This log is stored in*HADOOP_LOG_DIR/fairscheduler* **only.** Boolean value, default: false.
mapred.fairscheduler.dump.interval	If using the event log, this is the interval at which to dump complete scheduler state (list of pools and jobs) to the log. **NO** **only.** Integer value in milliseconds, default: 10000.

Allocation File (fair-scheduler.xml)

The allocation file configures minimum shares, running job limits, weights and preemption timeouts for each pool. Only users/pools whose values differ from the defaults need to be explicitly configured in this file. The allocation file is located in *HADOOP_HOME/conf/fair-scheduler.xml*. It can contain the following types of elements:

- *pool* elements, which configure each pool. These may contain the following sub-elements:
  - *minMaps* and *minReduces*, to set the pool's minimum share of task slots.
  - *maxMaps* and *maxReduces*, to set the pool's maximum concurrent task slots.
  - *schedulingMode*, the pool's internal scheduling mode, which can be *fair* for fair sharing or *fifo* for first-in-first-out.
  - *maxRunningJobs*, to limit the number of jobs from the pool to run at once (defaults to infinite).
  - *weight*, to share the cluster non-proportionally with other pools. For example, a pool with weight 2.0 will get a 2x higher share than other pools. The default weight is 1.0.
  - *minSharePreemptionTimeout*, the number of seconds the pool will wait before killing other pools' tasks if it is below its minimum share (defaults to infinite).
- *user* elements, which may contain a *maxRunningJobs* element to limit jobs. Note that by default, there is a pool for each user, so per-user limits are not necessary.
- *poolMaxJobsDefault*, which sets the default running job limit for any pools whose limit is not specified.

- *userMaxJobsDefault*, which sets the default running job limit for any users whose limit is not specified.
- *defaultMinSharePreemptionTimeout*, which sets the default minimum share preemption timeout for any pools where it is not specified.
- *fairSharePreemptionTimeout*, which sets the preemption timeout used when jobs are below half their fair share.
- *defaultPoolSchedulingMode*, which sets the default scheduling mode (*fair* or *fifo*) for pools whose mode is not specified.

Pool and user elements only required if you are setting non-default values for the pool/user. That is, you do not need to declare all users and all pools in your config file before running the fair scheduler. If a user or pool is not listed in the config file, the default values for limits, preemption timeouts, etc will be used.

An example allocation file is given below :

```
<?xml version="1.0"?>
<allocations>
<pool name="sample_pool">
<minMaps>5</minMaps>
<minReduces>5</minReduces>
<maxMaps>25</maxMaps>
<maxReduces>25</maxReduces>
<minSharePreemptionTimeout>300</minSharePreemptionTimeout>
</pool>
<user name="sample_user">
<maxRunningJobs>6</maxRunningJobs>
</user>
<userMaxJobsDefault>3</userMaxJobsDefault>
<fairSharePreemptionTimeout>600</fairSharePreemptionTimeout>
</allocations>
```

This example creates a pool sample_pool with a guarantee of 5 map slots and 5 reduce slots. The pool also has a minimum share preemption timeout of 300 seconds (5 minutes), meaning that if it does not get its guaranteed share within this time, it is allowed to kill tasks from other pools to achieve its share. The pool has a cap of 25 map and 25 reduce slots, which means that once 25 tasks are running, no more will be scheduled even if the pool's fair share is higher. The example also limits the number of running jobs per user to 3, except for sample_user, who can run 6 jobs concurrently. Finally, the example sets a fair share preemption timeout of 600 seconds (10 minutes). If a job is below half its fair share for 10 minutes, it will be allowed to kill tasks from other jobs to achieve its share. Note that the preemption settings require preemption to be enabled in *mapred-site.xml* as described earlier.

Any pool not defined in the allocation file will have no guaranteed capacity and no preemption timeout. Also, any pool or user with no max running jobs set in the file will be allowed to run an unlimited number of jobs.

Access Control Lists (ACLs)

The fair scheduler can be used in tandem with the "queue" based access control system in MapReduce to restrict which pools each user can access. To do this, first enable ACLs and set up some queues as described in the MapReduce usage guide, then set the fair scheduler to use one pool per queue by adding the following property in *HADOOP_CONF_DIR/mapred-site.xml*:

```
<property>
<name>mapred.fairscheduler.poolnameproperty</name>
<value>mapred.job.queue.name</value>
</property>
```

You can then set the minimum share, weight, and internal scheduling mode for each pool as described earlier. In addition, make sure that users submit jobs to the right queue by setting the *mapred.job.queue.name* property in their jobs.

Administration

The fair scheduler provides support for administration at runtime through two mechanisms:

1. It is possible to modify minimum shares, limits, weights, preemption timeouts and pool scheduling modes at runtime by editing the allocation file. The scheduler will reload this file 10-15 seconds after it sees that it was modified.

2. Current jobs, pools, and fair shares can be examined through the JobTracker's web interface, at *http://<JobTracker URL>/scheduler*. On this interface, it is also possible to modify jobs' priorities or move jobs from one pool to another and see the effects on the fair shares (this requires JavaScript).

The following fields can be seen for each job on the web interface:

- *Submitted* - Date and time job was submitted.
- *JobID, User, Name* - Job identifiers as on the standard web UI.
- *Pool* - Current pool of job. Select another value to move job to another pool.
- *Priority* - Current priority. Select another value to change the job's priority
- *Maps/Reduces Finished*: Number of tasks finished / total tasks.
- *Maps/Reduces Running*: Tasks currently running.
- *Map/Reduce Fair Share*: The average number of task slots that this job should have at any given time according to fair sharing. The actual number of tasks will go up and down depending on how much compute time the job has had, but on average it will get its fair share amount.

In addition, it is possible to view an "advanced" version of the web UI by going to *http://<JobTracker URL>/scheduler?advanced*. This view shows two more columns:

- *Maps/Reduce Weight*: Weight of the job in the fair sharing calculations. This depends on priority and potentially also on job size and job age if the*sizebasedweight* and *NewJobWeightBooster* are enabled.

Metrics

The fair scheduler can export metrics using the Hadoop metrics interface. This can be enabled by adding an entry to hadoop-metrics.properties to enable the fairscheduler metrics context. For example, to simply retain the metrics in memory so they may be viewed in the /metrics servlet:

```
fairscheduler.class=org.apache.hadoop.metrics.spi.NoEmitMetricsContext
```

Metrics are generated for each pool and job, and contain the same information that is visible on the /scheduler web page.

Capacity Schedular:

Overview

The CapacityScheduler is designed to run Hadoop Map-Reduce as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster while running Map-Reduce applications.

Traditionally each organization has it own private set of compute resources that have sufficient capacity to meet the organization's SLA under peak or near peak conditions. This generally leads to poor average utilization and the overhead of managing multiple independent clusters, one per each organization. Sharing clusters between organizations is a cost-effective manner of running large Hadoop installations since this allows them to reap benefits of economies of scale without creating private clusters. However, organizations are concerned about sharing a cluster because they are worried about others using the resources that are critical for their SLAs.

The CapacityScheduler is designed to allow sharing a large cluster while giving each organization a minimum capacity guarantee. The central idea is that the available resources in the Hadoop Map-Reduce cluster are partitioned among multiple organizations who collectively fund the cluster based on computing needs. There is an added benefit that an organization can access any excess capacity no being used by others. This provides elasticity for the organizations in a cost-effective manner.

Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rouge job or user. The CapacityScheduler provides a stringent set of limits to ensure that a single job or user or queue cannot consume dispropotionate amount of resources in the cluster. Also, the JobTracker of the cluster, in particular, is a precious resource and the CapacityScheduler provides limits on initialized/pending tasks and jobs from a single user and queue to ensure fairness and stability of the cluster.

The primary abstraction provided by the CapacityScheduler is the concept of *queues*. These queues are typically setup by administrators to reflect the economics of the shared cluster.

Features

The CapacityScheduler supports the following features:

- Capacity Guarantees - Support for multiple queues, where a job is submitted to a queue.Queues are allocated a fraction of the capacity of the grid in the sense that a certain capacity of resources will be at their disposal. All jobs submitted to a queue will have access to the capacity allocated to the queue. Adminstrators can configure soft limits and optional hard limits on the capacity allocated to each queue.
- Security - Each queue has strict ACLs which controls which users can submit jobs to individual queues. Also, there are safe-guards to ensure that users cannot view and/or modify jobs from other users if so desired. Also, per-queue and system administrator roles are supported.
- Elasticity - Free resources can be allocated to any queue beyond it's capacity. When there is demand for these resources from queues running below capacity at a future point in time, as tasks scheduled on these resources complete, they will be assigned to jobs on queues running below the capacity. This ensures that resources are available in a predictable and elastic manner to queues, thus preventing artifical silos of resources in the cluster which helps utilization.
- Multi-tenancy - Comprehensive set of limits are provided to prevent a single job, user and queue from monpolizing resources of the queue or the cluster as a whole to ensure that the system, particularly the JobTracker, isn't overwhelmed by too many tasks or jobs.
- Operability - The queue definitions and properties can be changed, at runtime, by administrators in a secure manner to minimize disruption to users. Also, a console is provided for users and administrators to view current allocation of resources to various queues in the system.
- Resource-based Scheduling - Support for resource-intensive jobs, wherein a job can optionally specify higher resource-requirements than the default, there-by accomodating applications with differing resource requirements. Currently, memory is the the resource requirement supported.
- Job Priorities - Queues optionally support job priorities (disabled by default). Within a queue, jobs with higher priority will have access to the queue's resources before jobs with lower priority. However, once a job is running, it will not be preempted for a higher priority job, *premption* is on the roadmap is currently not supported.

Installation

The CapacityScheduler is available as a JAR file in the Hadoop tarball under the *contrib/capacity-scheduler* directory. The name of the JAR file would be on the lines of hadoop-capacity-scheduler-*.jar.

You can also build the Scheduler from source by executing *ant package*, in which case it would be available under *build/contrib/capacity-scheduler*.

To run the CapacityScheduler in your Hadoop installation, you need to put it on the *CLASSPATH*. The easiest way is to copy the hadoop-capacity-scheduler-*.jar from to HADOOP_HOME/lib. Alternatively, you can modify *HADOOP_CLASSPATH* to include this jar, in conf/hadoop-env.sh.

Configuration

Using the CapacityScheduler

To make the Hadoop framework use the CapacityScheduler, set up the following property in the site configuration:

Property	Value
mapred.jobtracker.taskScheduler	org.apache.hadoop.mapred.CapacityTaskScheduler

Setting up queues

You can define multiple queues to which users can submit jobs with the CapacityScheduler. To define multiple queues, you should use the*mapred.queue.names* property in conf/hadoop-site.xml.

The CapacityScheduler can be configured with several properties for each queue that control the behavior of the Scheduler. This configuration is in the*conf/capacity-scheduler.xml.*

You can also configure ACLs for controlling which users or groups have access to the queues in conf/mapred-queue-acls.xml.

For more details, refer to Cluster Setup documentation.

Queue properties

Resource allocation

The properties defined for resource allocations to queues and their descriptions are listed in below:

Name	Description
mapred.capacity-scheduler.queue.<queue-name>.capacity	Percentage of the number of slots in the cluster that are made to be available for jobs in this queue. The sum of capacities for a
mapred.capacity-scheduler.queue.<queue-name>.maximum-	maximum-capacity defines a limit beyond which a queue cannot use the capacity of the cluster.This provides a means to limit how default, there is no limit. The maximum-capacity of a queue can only be greater than or equal to its minimum capacity. Default val capacity of the cluster. This property could be to curtail certain jobs which are long running in nature from occupying more than the absence of pre-emption, could lead to capacity guarantees of other queues being affected. One important thing to note is tha

capacity	based on the cluster's capacity it would change. So if large no of nodes or racks get added to the cluster , maximum Capacity in c
mapred.capacity-scheduler.queue.<queue-name>.minimum-user-limit-percent	Each queue enforces a limit on the percentage of resources allocated to a user at any given time, if there is competition for then and maximum value. The former depends on the number of users who have submitted jobs, and the latter is set to this property v property is 25. If two users have submitted jobs to a queue, no single user can use more than 50% of the queue resources. If a t more than 33% of the queue resources. With 4 or more users, no user can use more than 25% of the queue's resources. A value o
mapred.capacity-scheduler.queue.<queue-name>.user-limit-factor	The multiple of the queue capacity which can be configured to allow a single user to acquire more slots. By default this is set to 1 more than the queue's configured capacity irrespective of how idle th cluster is.
mapred.capacity-scheduler.queue.<queue-name>.supports-priority	If true, priorities of jobs will be taken into account in scheduling decisions.

Job initialization

Capacity scheduler lazily initializes the jobs before they are scheduled, for reducing the memory footprint on jobtracker. Following are the parameters, by which you can control the initialization of jobs per-queue.

Name	Description
mapred.capacity-scheduler.maximum-system-jobs	Maximum number of jobs in the system which can be initialized, concurrently, by the CapacityScheduler. are directly proportional to their queue capacities.
mapred.capacity-scheduler.queue.<queue-name>.maximum-initialized-active-tasks	The maximum number of tasks, across all jobs in the queue, which can be initialized concurrently. Once t will be queued on disk.
mapred.capacity-scheduler.queue.<queue-name>.maximum-initialized-active-tasks-per-user	The maximum number of tasks per-user, across all the of the user's jobs in the queue, which can be initi exceed this limit they will be queued on disk.
mapred.capacity-scheduler.queue.<queue-name>.init-accept-jobs-factor	The multipe of (maximum-system-jobs * queue-capacity) used to determine the number of jobs which are value is 10. If number of jobs submitted to the queue exceeds this limit, job submission are rejected.

Resource based scheduling   http://hadooptraininginhyderabad.co.in

The CapacityScheduler supports scheduling of tasks on a TaskTracker(TT) based on a job's memory requirements in terms of RAM and Virtual Memory (VMEM) on the TT node. A TT is conceptually composed of a fixed number of map and reduce slots with fixed slot size across the cluster. A job can ask for one or more slots for each of its component map and/or reduce slots. If a task consumes more memory than configured the TT forcibly kills the task.

Currently the memory based scheduling is only supported in Linux platform.

Additional scheduler-based config parameters are as follows:

Name	Description
mapred.cluster.map.memory.mb	The size, in terms of virtual memory, of a single map slot in the Map-Reduce framework, used by the scheduler. A job can ask for mu via mapred.job.map.memory.mb, upto the limit specified bymapred.cluster.max.map.memory.mb, if the scheduler supports the feature turned off.
mapred.cluster.reduce.memory.mb	The size, in terms of virtual memory, of a single reduce slot in the Map-Reduce framework, used by the scheduler. A job can ask for via mapred.job.reduce.memory.mb, upto the limit specified bymapred.cluster.max.reduce.memory.mb, if the scheduler supports the f feature is turned off.
mapred.cluster.max.map.memory.mb	The maximum size, in terms of virtual memory, of a single map task launched by the Map-Reduce framework, used by the scheduler. A map task via mapred.job.map.memory.mb, upto the limit specified by mapred.cluster.max.map.memory.mb, if the scheduler supports t feature is turned off.
mapred.cluster.max.reduce.memory.mb	The maximum size, in terms of virtual memory, of a single reduce task launched by the Map-Reduce framework, used by the schedule reduce task via mapred.job.reduce.memory.mb, upto the limit specified by mapred.cluster.max.reduce.memory.mb, if the scheduler su that this feature is turned off.
mapred.job.map.memory.mb	The size, in terms of virtual memory, of a single map task for the job. A job can ask for multiple slots for a single map task, rounded of mapred.cluster.map.memory.mb and upto the limit specified bymapred.cluster.max.map.memory.mb, if the scheduler supports the f feature is turned off iff mapred.cluster.map.memory.mb is also turned off (-1).
mapred.job.reduce.memory.mb	The size, in terms of virtual memory, of a single reduce task for the job. A job can ask for multiple slots for a single reduce task, rou of mapred.cluster.reduce.memory.mb and upto the limit specified bymapred.cluster.max.reduce.memory.mb, if the scheduler supports this feature is turned off iff mapred.cluster.reduce.memory.mb is also turned off (-1).

Reviewing the configuration of the CapacityScheduler

Once the installation and configuration is completed, you can review it after starting the MapReduce cluster from the admin UI.

- Start the MapReduce cluster as usual.
- Open the JobTracker web UI.
- The queues you have configured should be listed under the *Scheduling Information* section of the page.
- The properties for the queues should be visible in the *Scheduling Information* column against each queue.
- The /scheduler web-page should show the resource usages of individual queues.

Example

Here is a practical example for using CapacityScheduler:

```
<?xml version="1.0"?>
<configuration>
<!-- system limit, across all queues -->
<property>
  <name>mapred.capacity-scheduler.maximum-system-jobs</name>
  <value>3000</value>
<description>Maximum number of jobs in the system which can be initialized,
concurrently, by the CapacityScheduler.
</description>
</property>

<!-- queue: queueA -->
<property>
  <name>mapred.capacity-scheduler.queue.queueA.capacity</name>
  <value>8</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.user-limit-factor</name>
  <value>10</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueA.init-accept-jobs-factor</name>
  <value>100</value>
</property>

<!-- queue: queueB -->
<property>
  <name>mapred.capacity-scheduler.queue.queueB.capacity</name>
  <value>2</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueB.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueB.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueB.user-limit-factor</name>
  <value>1</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueB.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueB.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
   <name>mapred.capacity-scheduler.queue.queueB.init-accept-jobs-factor</name>
  <value>10</value>
</property>
```

```
<!-- queue: queueC -->
<property>
  <name>mapred.capacity-scheduler.queue.queueC.capacity</name>
  <value>30</value>
</property>
<property>


  <name>mapred.capacity-scheduler.queue.queueC.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueC.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueC.user-limit-factor</name>
  <value>1</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueC.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueC.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueC.init-accept-jobs-factor</name>
  <value>10</value>
</property>

<!-- queue: queueD -->
<property>
  <name>mapred.capacity-scheduler.queue.queueD.capacity</name>
  <value>1</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.user-limit-factor</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueD.init-accept-jobs-factor</name>
  <value>10</value>
</property>

<!-- queue: queueE -->
<property>
  <name>mapred.capacity-scheduler.queue.queueE.capacity</name>
  <value>31</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.user-limit-factor</name>
  <value>1</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueE.init-accept-jobs-factor</name>
  <value>10</value>
</property>

<!-- queue: queueF -->
<property>
  <name>mapred.capacity-scheduler.queue.queueF.capacity</name>
  <value>28</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.supports-priority</name>
  <value>false</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.minimum-user-limit-percent</name>
  <value>20</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.user-limit-factor</name>
  <value>1</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.maximum-initialized-active-tasks</name>
  <value>200000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.maximum-initialized-active-tasks-per-user</name>
  <value>100000</value>
</property>
<property>
  <name>mapred.capacity-scheduler.queue.queueF.init-accept-jobs-factor</name>
  <value>10</value>
</property>

</configuration>
```